BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

AR

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 17/14

A1

(11) International Publication Number: **WO 97/14105**

(43) International Publication Date: 17 April 1997 (17.04.97)

(21) International Application Number: PCT/US96/16092

(22) International Filing Date: 8 October 1996 (08.10.96)

(30) Priority Data:
08/541,602 10 October 1995 (10.10.95) US

(71) Applicant: ARCH DEVELOPMENT CORPORATION [US/US]; 1101 East 58th Street, Chicago, IL 60637 (US).

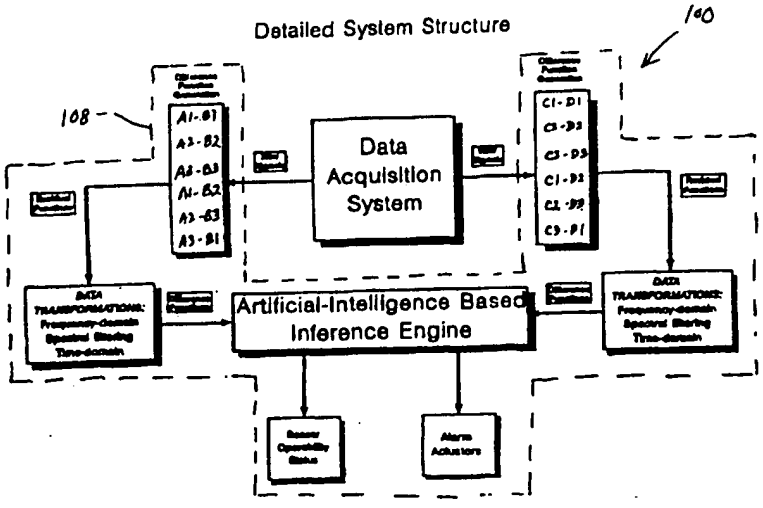
(72) Inventors: GROSS, Kenneth, C.; 1433 Carriage Lane, Bolingbrook, IL 60440-1437 (US). SINGER, Ralph, M.; 1602 Brestal Court, Naperville, Il 60565-2781 (US).

(74) Agents: RECHTIN, Michael, D. et al.; Reinhart, Boerner, Van Deuren, Norris & Rieselbach, s.c., Suite 2100, 1000 North Water Street, P.O. Box 92900, Milwaukee, WI 53202-0900 (US).

(81) Designated States: AU, CA, JP, KR, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: AN EXPERT SYSTEM FOR TESTING INDUSTRIAL PROCESSES AND DETERMINING SENSOR STATUS



(57) Abstract

A method and system (110) for monitoring both an industrial process and a sensor (104). The method and system include determining a minimum number of sensor pairs needed to test the industrial process as well as the sensor (104) for evaluating the state of operation of both. After obtaining two signals associated with one physical variable, a difference function is obtained by determining the arithmetic difference between the pair of signals over time. A frequency domain transformation is made of the difference function to obtain Fourier modes describing a composite function. A residual function is obtained by subtracting the composite function from the difference function and the residual function (free of nonwhite noise) is analyzed by a statistical probability ratio test.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	GB	United Kingdom	MW	Malawi
AT	Austria	GE	Georgia	MX	Mexico
AU	Australia	GN	Guinea	NE	Niger
BB	Barbados	GR	Greece	NL	Netherlands
BE	Belgium	HU	Hungary	NO	Norway
BF	Burkina Faso	IE	Ireland	NZ	New Zealand
BG	Bulgaria	IT	Italy	PL	Poland
BJ	Benin	JP	Japan	PT	Portugal
BR	Brazil	KE	Kenya	RO	Romania
BY	Belarus	KG	Kyrgystan	RU	Russian Federation
CA	Canada	KP	Democratic People's Republic of Korea	SD	Sudan
CF	Central African Republic			SE	Sweden
CG	Congo	KR	Republic of Korea	SG	Singapore
CH	Switzerland	KZ	Kazakhstan	SI	Slovenia
CI	Côte d'Ivoire	LI	Liechtenstein	SK	Slovakia
CM	Cameroon	LK	Sri Lanka	SN	Senegal
CN	China	LR	Liberia	SZ	Swaziland
CS	Czechoslovakia	LT	Lithuania	TD	Chad
CZ	Czech Republic	LU	Luxembourg	TG	Togo
DE	Germany	LV	Latvia	TJ	Tajikistan
DK	Denmark	MC	Monaco	TT	Trinidad and Tobago
EE	Estonia	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	UG	Uganda
FI	Finland	ML	Mali	US	United States of America
FR	France	MN	Mongolia	UZ	Uzbekistan
GA	Gabon	MR	Mauritania	VN	Viet Nam

An Expert System for Testing Industrial Processes and Determining Sensor Status

The United States Government has rights in this invention pursuant to Contract W-31-109-ENG-38 between the U.S. Department of Energy and the University of Chicago.

The present invention is concerned generally with an expert system and method for reliably monitoring industrial processes using a set of sensors. More particularly, the invention is concerned with an expert system and method for establishing a network of industrial sensors for parallel monitoring of industrial devices. The expert system includes a network of highly sensitive pattern recognition modules for automated parameter surveillance using a sequential probability ratio test.

Conventional parameter-surveillance schemes are sensitive only to gross changes in the mean value of a process, or to large steps or spikes that exceed some threshold limit check. These conventional methods suffer from either large numbers of false alarms (if thresholds are set too close to normal operating levels) or a large number of missed (or delayed) alarms (if the thresholds are set too expansively). Moreover, most conventional methods cannot perceive the onset of a process disturbance or sensor deviation which gives rise to a signal below the threshold level for an alarm condition.

In another conventional monitoring method, the Sequential Probability Ratio Test ("SPRT") has found wide application as a signal validation tool in the nuclear reactor industry. Two features of the SPRT technique make it attractive for parameter surveillance and fault detection: (1) early annunciation of the onset of a disturbance in noisy process variables, and (2) the SPRT technique has user-specifiable false-alarm and missed-alarm probabilities. One important drawback of the SPRT technique that has limited its adaptation to a broader range of applications is the fact that its mathematical formalism is founded upon an assumption that the signals it is monitoring are purely Gaussian, independent (white noise) random variables.

It is therefore an object of the invention to provide an improved method and system for continuous evaluation and/or modification of industrial processes and/or sensors monitoring the processes.

It is also an object of the invention to provide an improved method and system for automatically configuring a set of sensors to monitor an industrial process.

It is another object of the invention to provide a novel method and system for statistically processing industrial process signals having virtually any form of noise signal.

It is a further object of the invention to provide an improved method and system employing identical pairs of sensors for obtaining redundant readings of physical processes from an industrial process.

It is still an additional object of the invention to provide a novel method and system utilizing a plurality of signal pairs to generate difference functions to be analyzed for alarm information.

It is still a further object of the invention to provide an improved method and system including a plurality of single sensors for each industrial device or process for providing a real signal characteristic of a process and further providing a predicted sensor signal allowing formation of a difference signal between the predicted and real signal for subsequent analysis.

It is also an object of the invention to provide a novel method and system wherein difference functions are formed from pairs of sensor signals operating in parallel to monitor a plurality of like industrial devices.

It is yet an additional object of the invention to provide an improved method and system utilizing variable and multiple pairs of sensors for determining both sensor degradation and industrial process status.

It is still another object of the invention to provide a novel method and system having a plurality of sensors analytically configured by an expert system to establish the minimum

array of coupled sensors needed to monitor each sensor as well as the industrial process or devices.

An expert system has been developed that continuously monitors digitized signals from a set of sensors which are measuring a variety of physical variables (e.g., temperature, pressure, radiation level, vibration level, etc.). The expert system employs a sensitive pattern-recognition technique, the sequential probability ratio test ("SPRT") technique for early annunciation of sensor operability degradation. A SPRT module can monitor output from two identical sensors and determine if the statistical quality of the noise associated with either signal begins to change. In applications involving two or more industrial devices operated in parallel and equipped with identical sensors, a SPRT module applied to pairs of sensors monitoring the same physical process on the respective devices will provide sensitive annunciation of any physical disturbance affecting one of the devices. If each industrial device had only one sensor, it would not be possible for the SPRT technique to distinguish between equipment degradation and degradation of the sensor itself. In this application the primary benefit of the SPRT method would derive from its very early annunciation of the onset of the disturbance. Having this valuable information, additional diagnosis can then be performed to check the performance and calibration status of the sensor and to identify the root-cause of the signal anomaly.

For cases where each industrial device is equipped with multiple, redundant sensors, one can apply SPRT modules to pairs of sensors on each individual device for sensor-operability verification. In this case the expert system provides not only early annunciation of the onset of a disturbance, but also can distinguish between equipment degradation and degradation of its sensors. Moreover, when the expert system determines that the cause of the discrepant signals is due to a degraded sensor, it can identify the specific sensor that has failed.

In a simple generic application involving a single industrial device equipped with triply-redundant sensors for measurement of two physical variables, the expert system first

identifies the minimum unique set of signal pairs that will be needed for the network of interacting SPRT modules. Further, the system can operate using two industrial devices working in parallel (e.g., jet engines, propeller drive motors on a ship, turbomachinery in an industrial plant, etc.). Again the expert system identifies the pair-wise sensor combinations that it uses subsequently in building the conditional branching hierarchy for the SPRT-module configuration.

Other objects, features, alternative forms and advantages of the present invention will be readily apparent from the following description of the preferred embodiments thereof, taken in conjunction with the accompanying drawings described below.

Brief Description of the Drawings

FIGURE 1 illustrates the specified output of a pump's power output over time;

FIGURE 2 shows a Fourier composite curve generated using the pump spectral output of FIG. 1;

FIGURE 3 illustrates a residual function characteristic of the difference between FIGS. 1 and 2;

FIGURE 4A shows a periodogram of the spectral data of FIG. 1 and FIG. 4B shows a periodogram of the residual function of FIG. 3;

FIGURE 5A illustrates a noise histogram for the pump power output of FIG. 1 and FIG. 5B illustrates a noise histogram for the residual function of FIG. 3;

FIGURE 6A shows an unmodified delayed neutron detector signal from a first sensor and FIG. 6B is for a second neutron sensor; FIG. 6C shows a difference function characteristic of the difference between data in FIG. 6A and 6B and FIG. 6D shows the data output from a SPRT analysis with alarm conditions indicated by the diamond symbols;

FIGURE 7A illustrates an unmodified delayed neutron detector signal from a first sensor and FIG. 7B is for a second neutron sensor; FIG. 7C shows a difference function for the difference between the data of FIG. 7A and 7B and FIG. 7D shows the result of using the instant invention to modify the difference function to provide data free of serially correlated

noise to the SPRT analysis to generate alarm information and with alarm conditions indicated by the diamond signals;

FIGURE 8A and B illustrate a schematic functional flow diagram of the invention with FIG. 8A showing a first phase of the method of the invention and FIG. 8B shows the application of the method of the invention;

FIGURE 9 illustrates a plurality of sensors monitoring two physical variables of a single industrial device;

FIGURE 10 illustrates triply-redundant sensors monitoring two physical variables for two industrial devices;

FIGURE 11 illustrates an overall system structure employing the sensor array of FIG. 10;

FIGURE 12 illustrates triply-redundant sensors monitoring one physical variable for three industrial devices;

FIGURE 13 illustrates a logic diagram and conditional branching structure for an equipment surveillance module; and

FIGURE 14 illustrates a system of industrial devices and development of SPRT modules for monitoring the system.

Detailed Description of Preferred Embodiments

In a method of the invention signals from industrial process sensors can be used to annunciate, modify or terminate degrading or anomalous processes. The sensor signals are manipulated to provide input data to a statistical analysis technique, such as a process entitled Sequential Probability Ratio Test ("SPRT"). Details of this process and the invention therein are disclosed in S.N. 07/827,776 which is incorporated by reference herein in its entirety. A further illustration of the use of SPRT for analysis of data bases is set forth in U.S. Pat. No. 5,410,422 and copending application of the assignee SN 08/068,713, also incorporated by reference herein in their entirety. The procedures followed in a preferred methods are shown generally in FIG. 8 and also in FIG. 12. In performing such a preferred

analysis of the sensor signals, a dual transformation method is performed, insofar as it entails both a frequency-domain transformation of the original time-series data and a subsequent time-domain transformation of the resultant data. The data stream that passes through the dual frequency-domain, time-domain transformation is then processed with the SPRT procedure, which uses a log-likelihood ratio test. A computer software package, Appendix A, is also attached hereto covering the SPRT procedure and its implementation in the context of, and modified by, the instant invention.

In one preferred embodiment, successive data observations are performed on a discrete process Y, which represents a comparison of the stochastic components of physical processes monitored by a sensor, and most preferably pairs of sensors. In practice, the Y function is obtained by simply differencing the digitized signals from two respective sensors. Let y_k represent a sample from the process Y at time t_k. During normal operation with an undegraded physical system and with sensors that are functioning within specifications the y_k should be normally distributed with mean of zero. Note that if the two signals being compared do not have the same nominal mean values (due, for example, to differences in calibration), then the input signals will be pre-normalized to the same nominal mean values during initial operation.

In performing the monitoring of industrial processes, the system's purpose is to declare a first system, a second system, etc., degraded if the drift in Y is sufficiently large that the sequence of observations appears to be distributed about a mean +M or -M, where M is our pre-assigned system-disturbance magnitude. We would like to devise a quantitative framework that enables us to decide between two hypotheses, namely:

H_1 : Y is drawn from a Gaussian probability distribution function ("PDF") with mean M and variance σ^2.

H_2 : Y is drawn from a Gaussian PDF with mean 0 and variance σ^2.

We will suppose that if H_1 or H_2 is true, we wish to decide for H_1 or H_2 with probability $(1 - \beta)$ or $(1 - \alpha)$, respectively, where α and β represent the error (misidentification) probabilities.

From the conventional, well known theory of Wald, the test depends on the likelihood ratio l_n, where

$$l_n = \frac{\text{The probability of observed sequence } y_1, y_2 \ldots, y_n \text{ given } H_1 \text{ true}}{\text{The probability of observed sequence } y_1, y_2 \ldots, y_n \text{ given } H_2 \text{ true}} \quad (1)$$

After "n" observations have been made, the sequential probability ratio is just the product of the probability ratios for each step:

$$l_n = (PR_1) \bullet (PR_2) \bullet \ldots \bullet (PR_n) \quad (2)$$

or

$$l_n = \prod_{i=1}^{i=n} \frac{f(y_i|H_1)}{f(y_i|H_2)} \quad (3)$$

where $f(y|H)$ is the distribution of the random variable y.

Wald's theory operates as follows: Continue sampling as long as $A < l_n < B$. Stop sampling and decide H_1 as soon as $l_n \geq B$, and stop sampling and decide H_2 as soon as $l_n \leq A$. The acceptance thresholds are related to the error (misidentification) probabilities by the following expressions:

$$A = \frac{\beta}{1-\alpha}, \text{ and } B = \frac{1-\beta}{\alpha} \quad (4)$$

The (user specified) value of α is the probability of accepting H_1 when H_2 is true (false alarm probability). β is the probability of accepting H_2 when H_1 is true (missed alarm probability).

If we can assume that the random variable y_k is normally distributed, then the likelihood that H_1 is true (i.e., mean M, variance σ^2) is given by:

$$L(y_1, y_2..., y_n|H_1) = \frac{1}{(2\pi)^{n/2}\sigma^n} \exp\left[-\frac{1}{2\sigma^2}\left(\sum_{k=1}^{n} y_k^2 - 2\sum_{k=1}^{n} y_k M + \sum_{k=1}^{n} M^2\right)\right] \quad (5)$$

Similarly for H_2 (mean 0, variance σ^2):

$$L(y_1, y_2..., y_n|H_2) = \frac{1}{(2\pi)^{n/2}\sigma^n} \exp\left(-\frac{1}{2\sigma^2}\sum_{k=1}^{n} y_k^2\right) \quad (6)$$

The ratio of (5) and (6) gives the likelihood ratio l_n

$$l_n = \exp\left[-\frac{-1}{2\sigma^2}\sum_{k=1}^{n} M(M-2y_k)\right] \quad (7)$$

Combining (4) and (7), and taking natural logs gives

$$\ln\frac{\beta}{1-\alpha} < \frac{-1}{2\sigma^2}\sum_{k=1}^{n} M(M-2y_k) < \ln\frac{(1-\beta)}{\alpha} \quad (8)$$

Our sequential sampling and decision strategy can be concisely represented as:

If $l_n \le \ln\frac{\beta}{1-\alpha}$, Accept H_2 (9)

If $\ln\frac{\beta}{1-\alpha} < l_n < \ln\frac{1-\beta}{\alpha}$, Continue Sampling (10)

And if $l_n \ge \ln\frac{1-\beta}{\alpha}$, Accept H_1 (11)

Following Wald's sequential analysis, it is conventional that a decision test based on the log likelihood ratio has an optimal property; that is, for given probabilities α and β there is no other procedure with at least as low error probabilities or expected risk and with shorter length average sampling time.

A primary limitation that has heretofore precluded the applicability of Wald-type binary hypothesis tests for sensor and equipment surveillance strategies lies in the primary assumption upon which Wald's theory is predicated; i.e., that the original process Y is strictly "white" noise, independently-distributed random data. White noise is thus well known to be

a signal which is uncorrelated. Such white noise can, for example, include Gaussian noise. It is, however, very rare to find physical process variables associated with operating machinery that are not contaminated with serially-correlated, deterministic noise components. Serially correlated noise components are conventionally known to be signal data whose successive time point values are dependent on one another. Noise components include, for example, auto-correlated (also known as serially correlated) noise and Markov dependent noise. Auto-correlated noise is a known form of noise wherein pairs of correlation coefficients describe the time series correlation of various data signal values along the time series of data. That is, the data $U_1, U_2, \ldots, U_n$ have correlation coefficients $(U_1, U_2), (U_2, U_3), \ldots, (U_{n-1}, U_n)$ and likewise have correlation coefficients (U_1, U_3) (U_2, U_4), etc. If these data are auto-correlated, at least some of the coefficients are non-zero. Markov dependent noise, on the other hand, is a very special form of correlation between past and future data signals. Rather, given the value of U_k, the values of U_n, $n > k$, do not depend on the values of U_j where $j < k$. This implies the correlation pairs (U_j, U_n), given the value U_k, are all zero. If, however, the present value is imprecise, then the correlation coefficients may be nonzero. One form of this invention can overcome this limitation to conventional surveillance strategies by integrating the Wald sequential-test approach with a new dual transformation technique. This symbiotic combination of frequency-domain transformations and time-domain transformations produces a tractable solution to a particularly difficult problem that has plagued signal-processing specialists for many years.

In one preferred embodiment of the method shown in detail in FIG. 8, serially-correlated data signals from an industrial process can be rendered amenable to the SPRT testing methodology described hereinbefore. This is preferably done by performing a frequency-domain transformation of the original difference function Y. A particularly preferred method of such a frequency transformation is accomplished by generating a Fourier series using a set of highest "I" number of modes. Other procedures for rendering the data amenable to SPRT methods includes, for example, auto regressive techniques,

which can accomplish substantially similar results described herein for Fourier analysis. In the preferred approach of Fourier analysis to determine the "l" highest modes (see FIG. 8A):

$$Y_t = \frac{a_0}{2} + \sum_{m=1}^{\frac{N}{2}} (a_m \cos \omega_m t + b_m \sin \omega_m t) \qquad (12)$$

where $a_0/2$ is the mean value of the series, a_m and b_m are the Fourier coefficients corresponding to the Fourier frequency ω_m, and N is the total number of observations. Using the Fourier coefficients, we next generate a composite function, X_t, using the values of the largest harmonics identified in the Fourier transformation of Y_t. The following numerical approximation to the Fourier transform is useful in determining the Fourier coefficients a_m and b_m. Let x_j be the value of X_t at the jth time increment. Then assuming 2 π periodicity and letting

$\omega_m = 2\pi m/N$, the approximation to the Fourier transform yields:

$$a_m = \frac{2}{N} \sum_{j=0}^{N-1} x_j \cos \omega_m j \qquad b_m = \frac{2}{N} \sum_{j=0}^{N-1} x_j \sin \omega_m j \qquad (13)$$

for $0 < m < N/2$. Furthermore, the power spectral density ("PSD") function for the signal is given by l_m where

$$l_m = N \frac{a^2_m + b^2_m}{2} \qquad (14)$$

To keep the signal bandwidth as narrow as possible without distorting the PSD, no spectral windows or smoothing are used in our implementation of the frequency-domain transformation. In analysis of a pumping system of the EBR-II reactor of Argonne National Laboratory, the Fourier modes corresponding to the eight highest l_m provide the amplitudes and frequencies contained in X_t. In our investigations for the particular pumping system data taken, the highest eight l_m modes were found to give an accurate reconstruction of X_t while reducing most of the serial correlation for the physical variables studied. In other industrial processes, the analysis could result in more or fewer modes being needed to accurately construct the functional behavior of a composite curve. Therefore, the number of modes

used is a variable which is iterated to minimize the degree of nonwhite noise for any given application. As noted in FIG. 8A a variety of noise tests are applied in order to remove serially correlated noise.

The reconstruction of X_t uses the general form of Eqn. (12), where the coefficients and frequencies employed are those associated with the eight highest PSD values. This yields a Fourier composite curve (see end of flowchart in FIG. 8A) with essentially the same correlation structure and the same mean as Y_t. Finally, we generate a discrete residual function R_t by differencing corresponding values of Y_t and X_t. This residual function, which is substantially devoid of serially correlated contamination, is then processed with the SPRT technique described hereinbefore.

In a specific example application of the above referenced methodology, certain variables were monitored from the Argonne National Laboratory reactor EBR-II. In particular, EBR-II reactor coolant pumps (RCPs) and delayed neutron (DN) monitoring systems were tested continuously to demonstrate the power and utility of the invention. All data used in this investigation were recorded during full-power, steady state operation at EBR-II. The data have been digitized at a 2-per-second sampling rate using 2^{14} (16,384) observations for each signal of interest.

FIGS. 1-3 illustrate data associated with the preferred spectral filtering approach as applied to the EBR-II primary pump power signal, which measures the power (in kW) needed to operate the pump. The basic procedure of FIG. 8 was then followed in the analysis. FIG. 1 shows 136 minutes of the original signal as it was digitized at the 2-Hz sampling rate. FIG. 2 shows a Fourier composite constructed from the eight most prominent harmonics identified in the original signal. The residual function, obtained by subtracting the Fourier composite curve from the raw data, is shown in FIG. 3. Periodograms of the raw signal and the residual function have been computed and are plotted in FIG. 4. Note the presence of eight depressions in the periodogram of the residual function in FIG. 4B, corresponding to the most prominent periodicities in the original, unfiltered data.

Histograms computed from the raw signal and the residual function are plotted in FIG. 5. For each histogram shown we have superimposed a Gaussian curve (solid line) computed from a purely Gaussian distribution having the same mean and variance. Comparison of FIG. 5A and 5B provide a clear demonstration of the effectiveness of the spectral filtering in reducing asymmetry in the histogram. Quantitatively, this decreased asymmetry is reflected in a decrease in the skewness (or third moment of the noise) from 0.15 (raw signal) to 0.10 (residual function).

It should be noted here that selective spectral filtering, which we have designed to reduce the consequences of serial correlation in our sequential testing scheme, does not require that the degree of nonnormality in the data will also be reduced. For many of the signals we have investigated at EBR-II, the reduction in serial correlation is, however, accompanied by a reduction in the absolute value of the skewness for the residual function.

To quantitatively evaluate the improvement in whiteness effected by the spectral filtering method, we employ the conventional Fisher Kappa white noise test. For each time series we compute the Fisher Kappa statistic from the defining equation

$$\kappa = \left[\frac{1}{N}\sum_{k=1}^{N} I(\omega_k)\right]^{-1} I(L) \qquad (15)$$

where $I(\omega_k)$ is the PSD function (see Eq. 14) at discrete frequencies ω_k, and $I(L)$ signifies the largest PSD ordinate identified in the stationary time series.

The Kappa statistic is the ratio of the largest PSD ordinate for the signal to the average ordinate for a PSD computed from a signal contaminated with pure white noise. For EBR-II the power signal for the pump used in the present example has a κ of 1940 and 68.7 for the raw signal and the residual function, respectively. Thus, we can say that the spectral filtering procedure has reduced the degree of nonwhiteness in the signal by a factor of 28. Strictly speaking, the residual function is still not a pure white noise process. The 95% critical value for Kappa for a time series with 2^{14} observations is 12.6. This means that only for computed Kappa statistics lower than 12.6 could we accept the null hypothesis that the

signal is contaminated by pure white noise. The fact that our residual function is not purely white is reasonable on a physical basis because the complex interplay of mechanisms that influence the stochastic components of a physical process would not be expected to have a purely white correlation structure. The important point, however, is that the reduction in nonwhiteness effected by the spectral filtering procedure using only the highest eight harmonics in the raw signal has been found to preserve the pre-specified false alarm and missed alarm probabilities in the SPRT sequential testing procedure (see below). Table I summarizes the computed Fisher Kappa statistics for thirteen EBR-II plant signals that are used in the subject surveillance systems. In every case the table shows a substantial improvement in signal whiteness.

The complete SPRT technique integrates the spectral decomposition and filtering process steps described hereinbefore with the known SPRT binary hypothesis procedure. The process can be illustratively demonstrated by application of the SPRT technique to two redundant delayed neutron detectors (designated DND A and DND B) whose signals were archived during long-term normal (i.e., undegraded) operation with a steady DN source in EBR-II. For demonstration purposes a SPRT was designed with a false alarm rate, α, of 0.01. Although this value is higher than we would designate for a production surveillance system, it gives a reasonable frequency of false alarms so that asymptotic values of α can be obtained with only tens of thousands of discrete observations. According to the theory of the SPRT technique, it can be easily proved that for pure white noise (such as Gaussian), independently distributed processes, α provides an upper bound to the probability (per observation interval) of obtaining a false alarm--i.e., obtaining a "data disturbance" annunciation when, in fact, the signals under surveillance are undegraded.

FIGS. 6 and 7 illustrate sequences of SPRT results for raw DND signals and for spectrally-whitened DND signals, respectively. In FIGS. 6A and 6B, and 7A and 7B, respectively, are shown the DN signals from detectors DND-A and DND-B. The steady-state values of the signals have been normalized to zero.

TABLE I

Effectiveness of Spectral Filtering for Measured Plant Signals

	Fisher Kappa Test Statistic (N=16,384)	
Plant Variable I.D.	Raw Signal	Residual Function
Pump 1 Power	1940	68.7
Pump 2 Power	366	52.2
Pump 1 Speed	181	25.6
Pump 2 Speed	299	30.9
Pump 1 Radial Vibr (top)	123	67.7
Pump 2 Radial Vibr (top)	155	65.4
Pump 1 Radial Vibr (bottom)	1520	290.0
Pump 2 Radial Vibr (bottom)	1694	80.1
DN Monitor A	96	39.4
DN Monitor B	81	44.9
DN Detector 1	86	36.0
DN Detector 2	149	44.1
DN Detector 3	13	8.2

Normalization to adjust for differences in calibration factor or viewing geometry for redundant sensors does not affect the operability of the SPRT. FIGS. 6C and 7C in each figure show pointwise differences of signals DND-A and DND-B. It is this difference function that is input to the SPRT technique. Output from the SPRT method is shown for a 250-second segment in FIGS. 6D and 7D.

Interpretation of the SPRT output in FIGS. 6D and 7D is as follows: When the SPRT index reaches a lower threshold, A, one can conclude with a 99% confidence factor that there is no degradation in the sensors. For this demonstration A is equal to 4.60, which corresponds to false-alarm and missed-alarm probabilities of 0.01. As FIGS. 6D and 7D

illustrate, each time the SPRT output data reaches A, it is reset to zero and the surveillance continues.

If the SPRT index drifts in the positive direction and exceeds a positive threshold, B, of +4.60, then it can be concluded with a 99% confidence factor that there is degradation in at least one of the sensors. Any triggers of the positive threshold are signified with diamond symbols in FIGS. 6D and 7D. In this case, since we can certify that the sensors were functioning properly during the time period our signals were being archived, any triggers of the positive threshold are false alarms.

If we extend sufficiently the surveillance experiment illustrated in FIG. 6D, we can get an asymptotic estimate of the false alarm probability α. We have performed this exercise using 1000-observation windows, tracking the frequency of false alarm trips in each window, then repeating the procedure for a total of sixteen independent windows to get an estimate of the variance on this procedure for evaluating the false alarm probability. The resulting false-alarm frequency for the raw, unfiltered, signals is $\alpha = 0.07330$ with a variance of 0.000075. The very small variance shows that there would be only a negligible improvement in our estimate by extending the experiment to longer data streams. This value of α is significantly higher than the design value of $\alpha = 0.01$, and illustrates the danger of blindly applying a SPRT test technique to signals that may be contaminated by excessive serial correlation.

The data output shown in FIG. 7D employs the complete SPRT technique shown schematically in FIG. 8. When we repeat the foregoing exercise using 16 independent 1000-observation windows, we obtain an asymptotic cumulative false-alarm frequency of 0.009142 with a variance of 0.000036. This is less than (i.e., more conservative than) the design value of $\alpha = .01$, as desired.

It will be recalled from the description hereinbefore regarding one preferred embodiment, we have used the eight most prominent harmonics in the spectral filtration stage of the SPRT technique. By repeating the foregoing empirical procedure for evaluating

the asymptotic values of α, we have found that eight modes are sufficient for the input variables shown in Table I. Furthermore, by simulating subtle degradation in individual signals, we have found that the presence of serial correlation in raw signals gives rise to excessive missed-alarm probabilities as well. In this case spectral whitening is equally effective in ensuring that pre-specified missed-alarm probabilities are not exceeded using the SPRT technique.

In a different form of the invention, it is not necessary to have real sensors paired off to form a difference function. Each single sensor can provide a real signal characteristic of an ongoing process and a record artificial signal can be generated to allow formation of a difference function. Techniques such as an auto regressive moving average (ARMA) methodology can be used to provide the appropriate signal, such as a DC level signal, a cyclic signal or other predictable signal. Such an ARMA method is a well-known procedure for generating artificial signal values, and this method can even be used to learn the particular cyclic nature of a process being monitored enabling construction of the artificial signal.

The two signals, one a real sensor signal and the other an artificial signal, can thus be used in the same manner as described hereinbefore for two (paired) real sensor signals. The difference function Y is then formed, transformations performed and a residual function is determined which is free of serially correlated noise.

Fourier techniques are very effective in achieving a whitened signal for analysis, but there are other means to achieve substantially the same results using a different analytical methodology. For example, filtration of serial correlation can be accomplished by using the ARMA method. This ARMA technique estimates the specific correlation structure existing between sensor points of an industrial process and utilizes this correlation estimate to effectively filter the data sample being evaluated.

A technique has therefore been devised which integrates frequency-domain filtering with sequential testing methodology to provide a solution to a problem that is endemic to

industrial signal surveillance. The subject invention particularly allows sensing slow degradation that evolves over a long time period (gradual decalibration bias in a sensor, appearance of a new radiation source in the presence of a noisy background signal, wear out or buildup of a radial rub in rotating machinery, etc.). The system thus can alert the operator of the incipience or onset of the disturbance long before it would be apparent to visual inspection of strip chart or CRT signal traces, and well before conventional threshold limit checks would be tripped. This permits the operator to terminate, modify or avoid events that might otherwise challenge technical specification guidelines or availability goals. Thus, in many cases the operator can schedule corrective actions (sensor replacement or recalibration; component adjustment, alignment, or rebalancing; etc.) to be performed during a scheduled system outage.

Another important feature of the technique which distinguishes it from conventional methods is the built-in quantitative false-alarm and missed-alarm probabilities. This is quite important in the context of high-risk industrial processes and applications. The invention makes it possible to apply formal reliability analysis methods to an overall system comprising a network of interacting SPRT modules that are simultaneously monitoring a variety of plan variables. This amenability to formal reliability analysis methodology will, for example, greatly enhance the process of granting approval for nuclear-plant applications of the invention, a system that can potentially save a utility millions of dollars per year per reactor.

In another form of the invention, an artificial-intelligence based expert system 100 (see FIG. 12) has been developed for automatically configuring a set of sensors A, B, C and D to perform signal validation and sensor-operability surveillance in industrial applications that require high reliability, high sensitivity annunciation of degraded sensors, discrepant signals, or the onset of process anomalies. This expert system 100 comprises an interconnected network of high sensitivity pattern-recognition modules 102 (see FIGS. 9-11). The modules 102 embody the SPRT methodology described hereinbefore for automated

parameter surveillance. The SPRT method examines the noise characteristics of signals from identical pairs of sensors 104 deployed for redundant readings of continuous physical processes from a particular industrial device 106. The comparative analysis of the noise characteristics of a pair of signals, as opposed to their mean values, permits an early identification of a disturbance prior to significant (grossly observable) changes in the operating state of the process. As described in more detail hereinbefore, the SPRT method provides a superior surveillance tool because it is sensitive not only to disturbances in signal mean, but also to very subtle changes in the skewness, bias, or variance of the stochastic noise patterns associated with monitored signals. The use of two or more identical ones of the sensors 104 also permits the validation of these sensors 104, i.e., determines if the indicated disturbance is due to a change in the physical process or to a fault in either of the sensors 104.

For sudden, gross failures of one of the sensors 104 or components of the system 100, the SPRT module 102 would annunciate the disturbance as fast as a conventional threshold limit check. However, for slow degradation that evolves over a long time period (gradual decalibration bias in a sensor, wearout or buildup of a radial rub in rotating machinery, etc.), the SPRT module 102 provides the earliest possible annunciation of the onset of anomalous patterns in physical process variables. The SPRT-based expert system 100 can alert the operator to the incipience of the disturbance long before it would be apparent to visual inspection of strip chart or CRT signal traces, and well before conventional threshold limit checks would be tripped. This permits the operator to terminate or avoid events that might otherwise challenge technical specification guidelines or availability goals, and in many cases, to schedule corrective actions (sensor replacement or recalibration; component adjustment, alignment, or rebalancing; etc.) to be performed during a scheduled system outage.

The expert system 100 embodies the logic rules that convey to the operator the status of the sensors 104 and the industrial devices 106 connected to a SPRT network 108 (see

FIG. 12). When one or more of the SPRT modules 102 indicate a disturbance in sensor signals (i.e., the SPRT modules 102 trip) the expert system 100 determines which of the sensors 104 and/or the devices 106 are affected. The expert system 100 is designed to work with any network of the SPRT modules 102, encompassing any number of the industrial devices 106, process variables, and redundant ones of the sensors 104.

In a most preferred embodiment, the expert system 100 is operated using computer software written in the well known "LISP" language (see Appendix B attached hereto). In this embodiment, the expert system 100 is divided into two segments which act as pre- and post-processors for the SPRT module computer code. The pre-processor section is used to set up operation of the SPRT network 108 and forge connections between the sensor data stream and the SPRT modules 102. The post-processor section contains the logic rules which interpret the output of the SPRT modules 102.

The logic for the expert system 100 depends upon the grouping of the SPRT modules 102. Each of the SPRT modules 102 monitors two identical sensors 104 which measure a physical process variable. (See FIGS. 9-11.) The two sensors can either be redundant sensors 104 on one of the industrial devices 106 or separate sensors 104 on two identical ones of the industrial devices 106 that are operated in parallel. A group of the modules 102 entails all of the connections between the identical sensors 104 for a given physical variable on a group of the industrial devices 106. The number of the modules 102 in a group depends upon the number of identical devices 106 that are operated in parallel and the number of redundant sensors 104 on each device 106 which observe the response of a physical variable. For instance, suppose the expert system 100 is to be applied to an industrial system which contains three identical coolant pumps (not shown). Furthermore, suppose each coolant pump contains two redundant pressure transducers and one thermocouple (not shown). This system 100 would be modeled by two groups of the modules 102. The first group of the modules 102 would connect the six total pressure transducers which measure pump pressure. The second group of the modules 102 would

connect the three total thermocouples which measure coolant temperature. For a given group of related sensors 104, the data from each of the sensors 104 are fed into the modules 102. Since the module 102 performs a comparison test between the two sensors 104, the tripping of both of the modules 102 connected to the sensor 104 (in the absence of other tripped modules 102 in the same group) is a necessary and sufficient condition to conclude that the sensor 104 has failed. Therefore, for a group of related sensors 104, the minimum number of modules 102 needed to enable sensor detection is the same as the number of the sensors 104 in the group. For the example discussed above, the number of the modules 102 in the first group would be six, and the number of the modules 102 in the second group would be three.

In applications involving two or more identical ones of the industrial devices 106 operated in parallel and equipped with identical sensors 104, the module 102 applied to pairs of the sensors 104 monitoring the same physical process on the respective devices 106 will provide sensitive annunciation of any physical disturbance affecting one of the devices 106. If each of the devices 106 has only one of the sensors 104 though, it would not be possible for the expert system 100 to distinguish between device degradation and sensor degradation. In this case, the primary benefit of the method would derive from its very early annunciation of a disturbance. For cases in which each of the industrial devices 106 is equipped with multiple, redundant sensors 104, the modules 102 can be applied to pairs of the sensors 104 on each of the industrial devices 106 for sensor-operability verification. In this case, the expert system 100 not only provides early annunciation of a disturbance, but can also distinguish between device degradation and sensor degradation. Moreover, when the expert system 100 determines that the cause of the discrepant signals is due to a degraded one of the sensors 104, it can identify the specific sensor 104 that has failed.

FIG. 9 illustrates the first stage of the expert system 100 processing for a simple generic application involving a single one of the industrial devices 106 that is equipped with triply-redundant sensors 104 for measurement of two physical variables. The expert

system 100 first identifies the minimum unique set of signal pairs that will be needed for the network of interacting modules 102. FIG. 10 illustrates a generic application involving two of the industrial devices 106 that are operated in parallel. For this example, it is also assumed that triply-redundant sensors 104 are available for measuring each of two separate physical variables. Once again, the expert system 100 identifies the pair-wise sensor combinations that it uses in building the conditional branching hierarchy for the module configuration. FIG. 11 illustrates a generic application involving three industrial devices 106 that are operated in parallel. Triply-redundant sensors 104 for measuring one physical variable are assumed. The figure shows the pair-wise sensor combinations identified by the expert system 100 for building the conditional branching hierarchy. These figures also depict the three main branches for the logic rules contained in the expert system 100: a grouping of the modules 102 based on a single one of the industrial devices 106, two identical devices 106 operated in parallel or multiple (three or more) devices 106 operated in parallel. The expert system 100 however is not limited to only one of the three cases at a time. The industrial system 100 modeled can contain any number of independent single devices 106, doubly-redundant devices 106 and multiply-redundant devices 106. Each device group, in turn, may contain any number of redundant sensors 104 and any number of physical variables.

The expert system 100 is implemented using a stand-alone computer program set forth in the previously referenced Appendix B. In operation after the program initialization information has been gathered, the computer program prompts the user for the name of a data file that simulates the real-time behavior of the SPRT network 108 connected to the system 100 including the industrial devices 106. The SPRT data file contains space-delimited data values that represent the status of a corresponding module 102. The module 102 has two states: a 0 (non-tripped) state indicates that the signals from the sensors 104 monitored by the module 102 have not diverged from each other, while a 1 (tripped) state indicates that the signals from the sensors 104 monitored by the

module 102 have diverged from each other enough to be detected by the SPRT algorithm. Each line of data in the file represents the status of a group of related modules 102 at a given time. Each line contains a list of 0's and 1's that correspond to the state of all the modules 102 in the group. The number of groups in the network 108 depends upon the number of groups of identical devices 106 and the number of process variables monitored on each group of devices 106. If the network 108 contains more than one group of related modules 102, the data file will contain a corresponding number of lines to represent the status of all the modules 102 in the network 108 at a given time. For instance, if a system of the industrial devices 106 is modeled by four SPRT groups, the output file will contain four lines of SPRT data for each timestep in the simulation.

Execution of the program of Appendix B includes two procedures. The first procedure (SPRT_Expert) provides the instructions for the control of program execution and corresponds to the pre-processor section in an integrated SPRT expert system/SPRT module code. When executed, the procedure first prompts the user to specify the number of device groups in the application. A device group is a group of identical industrial devices 106 (one or more) that are operated in parallel and are equipped with redundant ones of the sensors 104. A device group can contain one or more physical variables. The program then prompts the user for the following information for each of the device groups:

(i) The name of the device group.

(ii) The number of physical variables in the device group.

(iii) The name of the first physical variable in the device group.

(iv) The number of redundant sensors 104 on each device 106 that measures the first physical variable.

If the device group contains more than one physical variable, the program will loop through steps (iii) and (iv) to obtain input data for the remaining physical variables. The number of

SPRT groups in the application is equal to the sum of the number of physical variables for each of the device groups. Namely,

$$N_{\text{SPRT Groups}} = \sum_{i=1}^{N_{\text{Device Groups}}} N_{\text{Physical Variables}_i}$$

Once the program has collected the required data to set up the system, it prompts the user for the name of the SPRT data file. Execution of the program consists of reading the SPRT status values from the data file and evaluating the status of the devices 106 and sensors 104 in the application, as inferred from the SPRT data. Program execution is controlled by a "do" loop. For each pass through the loop, the program reads the data which model the state of each of the SPRT modules 102 in the network at a given time. The SPRT data are then passed to the Analyze procedure. If any of the SPRT modules 102 in the application has tripped (i.e., SPRT value = 1), the Analyze procedure determines which device(s) 106 and/or sensor(s) 104) are affected and reports their status. Looping continues until the end of the data file is reached, upon which the program terminates.

The Analyze procedure contains the logic rules for the expert system 100. It corresponds to the post-processor section in an integrated SPRT expert system/SPRT module code. It is passed lists of 0's and 1's that represent the status of the SPRT modules 102 at any given timestep of the SPRT program. The number of lists passed to Analyze equals the number of SPRT groups. For each SPRT group, the procedure converts the SPRT data into a list of tripped SPRT modules 102. From the list of tripped SPRT modules 102, the status of the devices 106, and the sensors 104 modeled by the SPRT group are evaluated. Based on the number of devices 106 and the redundant sensors 104 in a SPRT group, the expert system 100 can determine which of the device(s) 106 and/or the sensors(s) 104 have failed. In some cases (e.g., if the number of the tripped modules 102 is one, or if the number of redundant sensors 104 in a group is one), the expert system 100 cannot conclude that the device 106 or the sensor 104 has failed, but can only signal that device or sensor failure is possible. Within Analyze, the logic rules are encapsulated by three procedures: SingleDevice, for a SPRT group applied to a single one of the industrial devices 106,

DualDevice, for a SPRT group applied to two parallely-operated industrial devices 106; and MultipleDevice, for a SPRT group applied to a group of three or more parallely-operated industrial devices 106.

The following nonlimiting example is illustrative of implementation of the expert system.

Example

The development of a network of the SPRT modules 102 for a general system 110 of the industrial devices 106 and the action of the corresponding logic rules are revealed by an example calculation. The system 110 of industrial devices 106 and a data file representing the transient behavior of the sensors 104 in the system 110 were created. A diagram of the system 110 is shown in FIG. 14 and contains two groups of the industrial devices 106. A first group 112 (identified as turbine devices) contains three of the identical devices 106. Each turbine is equipped with the sensors 104 to measure the steam temperature and steam pressure physical variables. There are two redundant sensors 104 on each turbine reading the steam temperature, while one of the sensors 104 measures the steam pressure. A second device group 114 consists of two coolant pumps. One physical variable, coolant flowrate, is gauged on each coolant pump by a group of four redundant sensors 104. The corresponding network of SPRT modules 102 for the system 110 is shown. Three groups of the SPRT modules 102 are required; with six of the modules 102 in a first module group for the steam temperature sensors on the turbines, three modules 102 in the second module group for the steam pressure sensors 104 on the turbines, and eight of the modules 102 in the third group of the modules 102 for the coolant flowrate sensors 104 on the coolant pumps.

A complete listing of the output from the test run follows hereinafter. Input entered by the user is identified by bold type. From the PC-Scheme prompt, the program is executed by first loading it into memory, and then calling the main procedure (SPRT_Expert). After displaying a title banner, the code asks the user to specify the number of device groups in the network.

[2] (load "SPRTEXPT.S")

OK

[3] (SPRT_Expert)

SPRT Expert System Simulation Program

Enter the number of device groups -> **2**

For each device group in the network, the program requests that the user supply the name of the devices in the group, the number of identical devices, the number of physical variables in the device group, and the names and numbers of redundant sensors for each physical variable. The input entered for device group #1 is:

DEVICE NAME:

Enter the name of device group number 1 -> **TURBINE**

DEVICE NUMBER:

Enter the number of devices in the TURBINE

device group -> **3**

PHYSICAL VARIABLE NUMBER:

Enter the number of physical variables in the TURBINE

device group -> **2**

PHYSICAL VARIABLE NAME:

Enter the name of physical variable number 1

of the TURBINE device group -> **STEAM TEMPERATURE**

SENSOR NUMBER:

Enter the number of redundant sensors for the STEAM TEMPERATURE

physical variable in the TURBINE device group -> **2**

PHYSICAL VARIABLE NAME:

Enter the name of physical variable number 2

of the TURBINE device group -> **STEAM PRESSURE**

SENSOR NUMBER:

Enter the number of redundant sensors for the STEAM PRESSURE

physical variable in the TURBINE device group -> **1**

The input entered for device group #2 is:

DEVICE NAME:

Enter the name of device group number 2 -> **COOLANT PUMP**

DEVICE NUMBER:

Enter the number of devices in the COOLANT PUMP

device group -> **2**

PHYSICAL VARIABLE NUMBER:

Enter the number of physical variables in the COOLANT PUMP

device group -> **1**

PHYSICAL VARIABLE NAME:

Enter the name of physical variable number 1

of the COOLANT PUMP device group -> **COOLANT FLOWRATE**

SENSOR NUMBER:

Enter the number of redundant sensors for the COOLANT FLOWRATE

physical variable in the COOLANT PUMP device group -> **4**

Once the input data for each device group have been obtained, the program displays a summary of the SPRT network, by identifying the SPRT groups in the network.

The number of SPRT Groups in the simulation is 3.

SPRT Group #1

contains 3 TURBINE industrial devices

with 2 STEAM TEMPERATURE redundant sensors.

SPRT Group #2

contains 3 TURBINE industrial devices

with 1 STEAM PRESSURE redundant sensor.

SPRT Group #3

contains 2 COOLANT PUMP industrial devices

with 4 COOLANT FLOWRATE redundant sensors.

The final input item required is the name of the data file containing the status values for the SPRT modules in the network.

Enter filename for SPRT data -> **TEST.DAT**

The analysis of the SPRT data is controlled by a do loop. For each pass through the loop, the program retrieves a line of data for each SPRT group in the network. Each block of data retrieved from the file represents the status of all SPRT modules in the network at a moment in time. The program analyzes each block of data to determine whether the SPRT status values imply device and/or sensor failures.

In the test calculation, the first block of data retrieved from the TEST.DAT file is:

0 0 0 0 0 0

0 0 0

0 0 0 0 0 0 0 0

Analyzing this data, the program reports that:

Analyzing SPRT data set number 1

SPRT Group #1:

No SPRTs have tripped.

SPRT Group #2:

No SPRTs have tripped.

SPRT Group #3:

No SPRTs have tripped.

The second block of data retrieved from the file contains some tripped SPRT modules:

1 0 0 0 0 0

0 0 0

0 1 0 0 0 0 0 0

Since only one module has tripped in SPRT groups #1 and #3, the program can conclude only that some device and/or sensor failures may have occurred. The program identifies which modules have tripped and which devices or sensors are affected.

Analyzing SPRT data set number 2

SPRT Group #1:

For the STEAM TEMPERATURE physical variable of the TURBINE devices:

These 1 of the 6 SPRTs have tripped ->

A1-B1

*** DEVICE NUMBER A OR DEVICE NUMBER B OF THE TURBINE DEVICES, SENSOR NUMBER A1, OR SENSOR NUMBER B1 MAY BE FAILING ***

SPRT Group #2:

No SPRTs have tripped.

SPRT Group #3:

For the COOLANT FLOWRATE physical variable of the COOLANT PUMP devices:

One SPRT has tripped -> A1-B2

*** ONE OR BOTH OF THE COOLANT PUMP DEVICES, SENSOR NUMBER A1 OR SENSOR NUMBER B2 MAY BE FAILING ***

In the third block of data, additional modules have tripped.

1 0 0 1 0 0

0 0 1

0 1 1 0 0 0 0 0

In SPRT group #1, two modules have tripped. Since both of the sensors on device A and both of the sensors on device B are affected, the code concludes that one of the two devices has failed. It cannot identify which of the devices has failed at this time though. In SPRT group #2, one module has tripped. The code concludes that one of the sensors or devices may be failing. Since both modules connected to sensor B2 in SPRT group #3 have tripped, the code concludes that sensor B3 has failed.

Analyzing SPRT data set number 3

SPRT Group #1:

For the STEAM TEMPERATURE physical variable of the TURBINE devices:

These 2 of the 6 SPRTs have tripped -

A1-B1 A2-B2

*** DEVICE NUMBER A OR DEVICE NUMBER B OF THE TURBINE DEVICES HAS FAILED ***

SPRT Group #2:

For the STEAM PRESSURE physical variable of the TURBINE devices:

These 1 of the 3 SPRTs have tripped -

C1-A1

*** DEVICE NUMBER C OR DEVICE NUMBER A OF THE TURBINE DEVICES, SENSOR NUMBER C1, OR SENSOR NUMBER A1 MAY BE FAILING ***

SPRT Group #3:

For the COOLANT FLOWRATE physical variable of the COOLANT PUMP devices:

These 2 of the 8 SPRTs have tripped -

A1-B2 A2-B2

*** SENSOR NUMBER B2 HAS FAILED ***

More modules have tripped in the fourth block of data.

1 0 1 1 0 0

0 1 1

0 1 1 0 0 1 0 0

Since three of the four modules connected to the sensors in device A of SPRT group #1 have tripped, the code concludes that device A has failed. In SPRT group #2, two of the three modules have tripped. But since there is only one sensor per device in this group, the code can only conclude that either a device or sensor failure has occurred. In SPRT group #3, three modules have tripped. The code concludes that one of the two devices has failed.

Analyzing SPRT data set number 4

SPRT Group #1:

For the STEAM TEMPERATURE physical variable of the TURBINE devices:

These 3 of the 6 SPRTs have tripped ->

A1-B1 C1-A1 A2-B2

*** DEVICE NUMBER A OF THE TURBINE DEVICES HAS FAILED ***

SPRT Group #2:

For the STEAM PRESSURE physical variable of the TURBINE devices:

These 2 of the 3 SPRTs have tripped ->

B1-C1 C1-A1

*** DEVICE NUMBER C OF THE TURBINE DEVICES,

OR SENSOR NUMBER C1 HAS FAILED ***

SPRT Group #3:

For the COOLANT FLOWRATE physical variable of the COOLANT PUMP devices:

These 3 of the 8 SPRTs have tripped ->

A1-B2 A2-B2 A3-B4

*** ONE OR BOTH OF THE COOLANT PUMP DEVICES HAVE FAILED ***

Notice that two of the tripped modules in SPRT group #3 implicate a failure of sensor B2, the conclusion reached by the analysis of the third block of data. But since an additional module has tripped, the code changes its conclusion from a sensor failure to a failure of one or both of the devices. Although the third module trip may be a spurious trip (i.e., the SPRT modules have a finite false alarm probability) which would mean that the earlier conclusion still holds, the code conservatively concludes that none of the trips are spurious and decides that a device failure has occurred. The code assumes that no module trip is spurious, which causes the code to consistently pick the most conservative conclusion when more than one conclusion can be deduced from the data.

The fifth and last set of data in the file contains additional module trips.

1 1 1 1 0 0

1 1 1

0 1 1 0 0 1 1 0

The additional trips in SPRT group #1 cause the code to conclude that more than one device in the group is affected. In SPRT group #2, all three modules in the group have tripped. Whenever all SPRT modules in a group trip, the code concludes that all devices in the group have failed. In SPRT group #3 the additional module trip does not change the conclusion, since the worst-case conclusion (i.e., one or both of the devices in the group have failed) for the group has already been reached.

Analyzing SPRT data set number 5

SPRT Group #1:

For the STEAM TEMPERATURE physical variable of the TURBINE devices:

These 4 of the 6 SPRTs have tripped ->

A1-B1 B1-C1 C1-A1 A2-B2

*** DEVICE NUMBER B OF THE TURBINE DEVICES HAS FAILED ***

*** SENSOR NUMBER C1 HAS FAILED ***

*** DEVICE NUMBER A OF THE TURBINE DEVICES HAS FAILED ***

SPRT Group #2:

For the STEAM PRESSURE physical variable of the TURBINE devices:

All 3 SPRTs have tripped

*** ALL 3 OF THE TURBINE DEVICES HAVE FAILED ***

SPRT Group #3:

For the COOLANT FLOWRATE physical variable of the COOLANT PUMP devices:

These 4 of the 8 SPRTs have tripped ->

A1-B2 A2-B2 A3-B4 A4-B4

*** ONE OR BOTH OF THE COOLANT PUMP DEVICES HAVE FAILED ***

Notice that the SPRT group #1, the code concludes that devices A and B have failed, but for device C it concludes that only one of its sensors has failed. For device groups containing multiple devices (i.e., three or more identical devices), the code applies its logic rules to each of the devices independently. Since for devices A and B both of their sensors are involved, the code concludes that the devices have failed. For device C only the first sensor is involved, thus the code concludes that only the first sensor on the device has failed. The code reaches the end of the file after analyzing the fifth block of data, causing the code to terminate. For SPRT group #3, a failure of one or both of the devices is indicated, although the pattern of tripped modules can also be interpreted as a simultaneous failure of sensors B2 and B4. The code indicates a device failure because concurrent failures of two or more sensors in a group is deemed to be highly improbable.

While preferred embodiments of the invention have been shown and described, it will be clear to those skilled in the art that various changes and modifications can be made without departing from the invention in its broader aspects as set forth in the claims provided hereinafter.

SOFTWARE APPENDIX A

```
#include <stdio.h>

main()
{

  FILE *falarm, *fopen();
  int fclose(),fprintf();
  static double thresh = 4.6;


  double sig1,sig2,sig3,sig4,sig5,sig6;
  double diff1,diff2,diff3,diff4,diff5,diff6,diff7;
  double sprt1,sprt2,sprt3,sprt4,sprt5,sprt6,sprt7,dumsprt;
  double p1,p2,p3,p4,p5,p6,p7,p0;
  int totalm = 0;
  int num = 0;
  int alm = 0;
  double v1,v2,v3,v4,v5,v6;
  double m1,m2,m3,m4,m5,m6;
  double sfm,q1,q2,q3,q4,q5,q6,q7;
  long seed1 = 13;
  long seed2 = 23;
  long seed3 = 27;
  long seed4 = 31;
  long seed5 = 53;
  long seed6 = 19;
  extern void normsim();
  extern void sprt();

/*    Get the parameters needed in the program                    */

  printf("\nInput the sensor failure magnitude SFM.");
  scanf("%lf",&sfm);
  printf("\nInput the variances of signals 1..6 separated by commas.");
  scanf("%lf,%lf,%lf,%lf,%lf,%lf",&v1,&v2,&v3,&v4,&v5,&v6);
  printf("\nInput the mean of signals 1..6 separated by commas.");
  scanf("%lf,%lf,%lf,%lf,%lf,%lf",&m1,&m2,&m3,&m4,&m5,&m6);


  falarm = fopen("alarms.dat","w");

  printf("\nPutting initial stuff in file.");
  fprintf(falarm,"\nEmpirical reliability - configuration 1");
  fprintf(falarm,"\n---------------------------------------");
  fprintf(falarm,"\nSensor Failure Magnitude SFM = %lf",sfm);
  fprintf(falarm,"\nSignal A1: mean mu=%lf  variance v=%lf",m1,v1);
  fprintf(falarm,"\nSignal A2: mean mu=%lf  variance v=%lf",m2,v2);
  fprintf(falarm,"\nSignal B1: mean mu=%lf  variance v=%lf",m3,v3);
  fprintf(falarm,"\nSignal B2: mean mu=%lf  variance v=%lf",m4,v4);
  fprintf(falarm,"\nSignal C1: mean mu=%lf  variance v=%lf",m5,v5);
  fprintf(falarm,"\nSignal C2: mean mu=%lf  variance v=%lf",m6,v6);
  fprintf(falarm,"\n \n \n \n");

  printf("\nStarting sprt.....");
  p1 = 0.0;p2 = 0.0;p3 = 0.0;p4 = 0.0;p5 = 0.0;p6 = 0.0;p7 = 0.0;p0 = 0.0;
  q1 = sfm/(v1+v2);q2 = sfm/(v3+v4);q3 = sfm/(v5+v6);
  q4 = sfm/(v1+v3);q5 = sfm/(v2+v4);q6 = sfm/(v3+v5);q7 = sfm/(v4+v6);
  while (num < 500)
    {
     num ++1;
     sprt1 = 0.0;
     sprt2 = 0.0;
     sprt3 = 0.0;
     sprt4 = 0.0;
```

```
sprt5 = 0.0;
sprt6 = 0.0;
sprt7 = 0.0;
dumsprt = 0.0;
alm = 0;
do
 {
 normsim(&sig1,&sig2,v1,v2,m1,m2,&seed1,&seed2);
 normsim(&sig3,&sig4,v3,v4,m3,m4,&seed3,&seed4);
 normsim(&sig5,&sig6,v5,v6,m5,m6,&seed5,&seed6);
 diff1 = sig1 - sig2;
 diff2 = sig3 - sig4;
 diff3 = sig5 - sig6;
 sprt(sfm,q1,thresh,diff1,&sprt1,&dumsprt);
 sprt(sfm,q2,thresh,diff2,&sprt2,&dumsprt);
 sprt(sfm,q3,thresh,diff3,&sprt3,&dumsprt);
} while((sprt1 < thresh) && (sprt2 < thresh) && (sprt3 < thresh));
if ((sprt1 == thresh) || (sprt2 == thresh) || (sprt3 == thresh))
  {
   printf("\nsprt1=%lf,sprt2=%lf,sprt3=%lf",sprt1,sprt2,sprt3);
  if (sprt1 == thresh)
     do
       {
       if ((sprt2 < thresh) && (sprt2 > -thresh))
         {
          normsim(&sig3,&sig4,v3,v4,m3,m4,&seed3,&seed4);
          diff2 = sig3 - sig4;
          sprt(sfm,q2,thresh,diff2,&sprt2,&dumsprt);
         }
       if ((sprt3 < thresh) && (sprt3 > -thresh))
         {
          normsim(&sig5,&sig6,v5,v6,m5,m6,&seed5,&seed6);
          diff3 = sig5 - sig6;
          sprt(sfm,q3,thresh,diff3,&sprt3,&dumsprt);
         }
      } while(((sprt2<thresh) && (sprt2>-thresh)) ||
                        ((sprt3<thresh) && (sprt3>-thresh)));
    else if (sprt2 == thresh)
     do
       {
        if ((sprt1 < thresh) && (sprt1 > -thresh))
          {
           normsim(&sig1,&sig2,v1,v2,m1,m2,&seed1,&seed2);
           diff1 = sig1 - sig2;
           sprt(sfm,q1,thresh,diff1,&sprt1,&dumsprt);
          }
        if ((sprt3 < thresh) && (sprt3 > -thresh))
          {
           normsim(&sig5,&sig6,v5,v6,m5,m6,&seed5,&seed6);
           diff3 = sig5 - sig6;
           sprt(sfm,q3,thresh,diff3,&sprt3,&dumsprt);
          }
      } while(((sprt1<thresh) && (sprt1>-thresh)) ||
                        ((sprt3<thresh) && (sprt3>-thresh)));
    else if (sprt3 == thresh)
      do
       {
        if ((sprt2 < thresh) && (sprt2 > -thresh))
          {
           normsim(&sig3,&sig4,v3,v4,m3,m4,&seed3,&seed4);
           diff2 = sig3 - sig4;
           sprt(sfm,q2,thresh,diff2,&sprt2,&dumsprt);
          }
        if ((sprt1 < thresh) && (sprt1 > -thresh))
          {
           normsim(&sig1,&sig2,v1,v2,m1,m2,&seed1,&seed2);
```

```
        diff1 = sig1 - sig2;
        sprt(sfm,q1,thresh,diff1,&sprt1,&dumsprt)
        sprt(sfm,q2,thresh,diff1,&sprt2,&dumsprt)
      }
    } while(((sprt2<thresh) && (sprt2>-thresh)) ||
                ((sprt1<thresh) && (sprt1>-thresh)));

printf("\nsprt1= %lf,sprt2= %lf,sprt3= %lf",sprt1,sprt2,sprt3);
if ((sprt1 == thresh) &&(sprt2 == -thresh) && (sprt3 == -thresh))
  {
   do
    {
     normsim(&sig1,&sig3,v1,v3,m1,m3,&seed1,&seed3);
     diff4 = sig1 - sig3;
     sprt(sfm,q4,thresh,diff4,&sprt4,&dumsprt);
    } while((sprt4 < thresh) && (sprt4 > -thresh));
    if (sprt4 < thresh)
      {
       do
         {
         normsim(&sig2,&sig4,v2,v4,m2,m4,&seed2,&seed4);
         diff5 = sig2 - sig4;
         sprt(sfm,q5,thresh,diff5,&sprt5,&dumsprt);
        } while ((sprt5 < thresh) && (sprt5 > -thresh));
        if (sprt5 == thresh) alm = 2;
       }
     else if (sprt4 == thresh) alm = 1;
    printf("\nsprt4=%lf,sprt5=%lf",sprt4,sprt5);
    printf("\nAlarm = %d",alm);
    }

  else if ((sprt2 == thresh) &&(sprt1 == -thresh) && (sprt3 == -thresh))
    {
     do
       {
       normsim(&sig1,&sig3,v1,v3,m1,m3,&seed1,&seed3);
       diff4 = sig3 - sig1;
       sprt(sfm,q4,thresh,diff4,&sprt4,&dumsprt);
      } while((sprt4 < thresh) && (sprt4 > -thresh));
      if (sprt4 < thresh)
        {
         do
           {
           normsim(&sig2,&sig4,v2,v4,m2,m4,&seed2,&seed4);
           diff5 = sig4 - sig2;
           sprt(sfm,q5,thresh,diff5,&sprt5,&dumsprt);
          } while ((sprt5 < thresh) && (sprt5 > -thresh));
          if (sprt5 == thresh) alm = 4;
         }
       else if (sprt4 == thresh) alm = 3;
     printf("\nsprt4=%lf,sprt5=%lf",sprt4,sprt5);
     printf("\nAlarm = %d",alm);
     }

    else if ((sprt3 == thresh) &&(sprt2 == -thresh) && (sprt1 == -thresh))
      {
       do
         {
         normsim(&sig3,&sig5,v3,v5,m3,m5,&seed3,&seed5);
         diff6 = sig5 - sig3;
         sprt(sfm,q6,thresh,diff6,&sprt6,&dumsprt);
        } while((sprt6 < thresh) && (sprt6 > -thresh));
        if (sprt6 < thresh)
          {
           do
             {
             normsim(&sig4,&sig6,v4,v6,m4,m6,&seed4,&seed5);
```

```
            sprt(sfm,q7   ch,diff7,&sprt7,&cumsprt);
          } while ((sprt  < thresh) && (sprt7 > -thres  ));
          if (sprt7 == thresh) alm = 6;
         }
       else if (sprt6 == thresh) alm = 5;
      printf("\nsprt6=%lf,sprt7=%lf",sprt6,sprt7);
      printf("\nAlarm = %d",alm);
      }
   else alm = 7;
   if (alm == 1) p1 += 1.0;
   if (alm == 2) p2 += 1.0;
   if (alm == 3) p3 += 1.0;
   if (alm == 4) p4 += 1.0;
   if (alm == 5) p5 += 1.0;
   if (alm == 6) p6 += 1.0;
   if (alm == 7) p7 += 1.0;
   totalm += 1;
  }
 }

p1 = p1/totalm;
p2 = p2/totalm;
p3 = p3/totalm;
p4 = p4/totalm;
p5 = p5/totalm;
p6 = p6/totalm;
p7 = p7/totalm;

fprintf(falarm,"\Empirical probabilities:");
fprintf(falarm,"\n  Total number of diagnostics run: %d",totalm);
fprintf(falarm,"\n  P(Alarm 1 sounded) = %lf",p1);
fprintf(falarm,"\n  P(Alarm 2 sounded) = %lf",p2);
fprintf(falarm,"\n  P(Alarm 3 sounded) = %lf",p3);
fprintf(falarm,"\n  P(Alarm 4 sounded) = %lf",p4);
fprintf(falarm,"\n  P(Alarm 5 sounded) = %lf",p5);
fprintf(falarm,"\n  P(Alarm 6 sounded) = %lf",p6);
fprintf(falarm,"\n  P(Alarm 7 sounded) = %lf",p7);

fclose(falarm);

}

*   In running this code for the nuclear plant data, two things need to be  ;
*   changed when changing days:  the filename and the title.  The filename   ;
*   The filename to be changed is at the top of the following code on the    ;
*   first line of code(DO NOT CHANGE THE DIRECTORY).  The title to be        ;
*   changed is on the 6th line of code.  To use the mean for the current     ;
*   day, leave all procedures in as they are.  To use the mean of the        ;
*   day ran previously, comment out the 7-9th lines of code.  This will      ;
*   cause the means of the data to be read from a file created at the        ;
*   last run - this way you can compare the whole month by using the mean    ;
*   of the first day.  Output is directed to the file 'sprtfp.lst'.  In      ;
*   this file, the results of the sprt run including longest sequence of     ;
*   failures and total number of decisions for each of the signal pairs      ;
*   is output.  The signal pairs are c1-c2, c3-c4, c5-c6, c7-c8, c9-c11,     ;
*   and c10-c12.  In addition to the cumulative results of the run, the      ;
*   SPRT1 and SPRT2 values at each data point are also given (to remove      ;
*   this feature, comment out the 'proc print' procedure on the last line    ;
*   of the macro).                                                           ;
*                  Written by Kristin K. Hoyer                               ;

filename sprtfil '-hoyer/data/fp/sprtvals.lst';
libname datlib '-hoyer/data/fp';
```

```
proc means mean noprint;
     var c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17
         c18 c19 c20 c21 c22;
     output out=datlib.fpmeans mean=m1 m2 m3 m4 m5 m6 m7 m8 m9 m10 m11 m12
         m13 m14 m15 m16 m17 m18 m19 m20 m21 m22;
data signal; set signal;
     i = 1;
     set datlib.fpmeans point=i;
     diff1 = c1-m1-(c2-m2);
     diff2 = c2-m2-(c3-m3);
     diff3 = c4-m4-(c5-m5);
     diff4 = c5-m5-(c6-m6);
     diff5 = c7-m7-(c8-m8);
     diff6 = c9-m9-(c10-m10);
     diff7 = c11-m11-(c12-m12);
     diff8 = c13-m13-(c14-m14);
     diff9 = c15-m15-(c16-m16);
     diff10 = c17-m17-(c18-m18);
     diff11 = c19-m19-(c20-m20);
     diff12 = c21-m21-(c22-m22);
     keep diff1 diff2 diff3 diff4 diff5 diff6 diff7 diff8 diff9
          diff10 diff11 diff12;
proc means data=signal var noprint;
     var diff1 diff2 diff3 diff4 diff5 diff6 diff7 diff8 diff9 diff10
         diff11 diff12;
     output out=datlib.fpvars var=v1 v2 v3 v4 v5 v6 v7 v8 v9 v10 v11 v12;
data signal; set signal;
     i=1;
     set datlib.fpvars point=i;
     sfm1 = 4.5*v1**0.5;
     sfm2 = 2.5*v2**0.5;
     sfm3 = 3.5*v3**0.5;
     sfm4 = 2.5*v4**0.5;
     sfm5 = 2.5*v5**0.5;
     sfm6 = 2.5*v6**0.5;
     sfm7 = 2.5*v7**0.5;
     sfm8 = 2.5*v8**0.5;
     sfm9 = 2.5*v9**0.5;
     sfm10 = 2.5*v10**0.5;
     sfm11 = 2.5*v11**0.5;
     sfm12 = 2.5*v12**0.5;
     output;
*****************************************************************;
*                                                               *;
*   This is the updated, working SPRT macro.                    *;
*   In order to use this macro, several                         *;
*   things need to be set up in the main program.               *;
*   Call a data step and retrieve the data, the                 *;
*   sensor failure magitude, and the variance of                *;
*   the signals.  The variable must be renamed                  *;
*   as follows: y = data, sfm=sensor failure magn.              *;
*   v=variance.  To change the threshhold limit                 *;
*   for the SPRT, change the value of the macro                 *;
*   variable thresh (in the SPRT macro) to the                  *;
*   desired value.                                              *;
*****************************************************************;

macro do2sprt
g=sfm/v;
thrsh = 6.9;
wndw = 50000;
if (_N_ = 1) then do;
   retain sprt1 0;retain sprt2 0;
   retain fail1 0;retain fail2 0;
   retain failpt 0;
```

```
   retain fail2pt 0;
   retain fseqpt 0;
   retain failseq 0;
   retain f1 0;
   retain f2 0;
   retain f1pt 0;
   retain f2pt 0;
end;
retain yp1;retain yp2;
INC1= -G*((STM/2)-Y)  ; INC2 = -G*((STM/2)-Y) ;
if (yp1>= thrsh or yp1<= -thrsh) then sprt1 = inc1;
   else sprt1 = yp1 +  inc1;
if (yp2>= thrsh or yp2<= -thrsh) then sprt2 = inc2;
   else sprt2 = yp2 +  inc2;
yp1 = sprt1;yp2 = sprt2;
IF SPRT1<=-thrsh THEN SPRT1=-thrsh ; IF SPRT2 <= -thrsh THEN SPRT2=-thrsh;
IF SPRT1>= thrsh THEN SPRT1=thrsh;
IF SPRT2>= thrsh THEN SPRT2=thrsh;
IF (SPRT1 >= thrsh) THEN TRIPH1 = 1;
IF (SPRT2 >= thrsh) THEN TRIPH1 = 1;
IF (SPRT1 <= -thrsh) THEN TRIPH2 = 1;
IF (SPRT2 <= -thrsh) THEN TRIPH2 = 1;
IF (SPRT1 <= -thrsh) THEN f1 = 0;
ELSE IF (SPRT1 >= thrsh) THEN DO;
        F1= fail1 + 1;
        IF (F1 = 1) THEN F1PT = _N_;
     END;
     ELSE DO;
         F1 = fail1;
         F1PT = fail1pt;
     END;
IF (SPRT2 <= -thrsh) THEN f2 = 0;
ELSE IF (SPRT2 >= thrsh) THEN DO;
       F2 = fail2 + 1;
       IF (F2 = 1) THEN F2PT = _N_;
     END;
     ELSE DO;
         F2 = fail2;
         F2pt = fail2pt;
     END;
IF (F1 >= FAILSEQ) OR (F2 >= FAILSEQ) THEN DO;
    IF (F1 >= F2) THEN DO;
              FAILSEQ = F1;FSEQPT = F1PT;END;
    ELSE DO;FAILSEQ = F2;FSEQPT = F2PT;END;
END;
ELSE DO;FAILSEQ = 0;FSEQPT = 0;END;
fail1 = f1;fail2 = f2;
fail1pt = f1pt;fail2pt = f2pt;
CNT + 1;
VAL = MOD(_N_,wndw);
IF VAL = 0 OR EOF THEN DO;
  FILE PRINT;
  IF (CNT > 0) THEN DO;
     FREQH1 = TRIPH1 / CNT;
     FREQH2 = TRIPH2 / CNT;
     TOTH1 + FREQH1;TOTH2 + FREQH2;TOTCNT + 1;
     TH1 = TOTH1 / TOTCNT; TH2 = TOTH2 / TOTCNT;
     ADDTRIP = FREQH1+FREQH2;ADDT=TH1+TH2;
     ALPH1=TOTH1/(TOTH1+TOTH2)*100.0; BET1 = TOTH2 / (TOTH1+TOTH2);
     PUT '          TRIPPING FREQUENCY- WINDOW SIZE:  ' CNT/
     '    H1 TRIPPING FREQUENCY / OBSERVATION(=>FAILURE):  ' FREQH1/
     '    H2 TRIPPING FREQUENCY / OBSERVATION(=>NORMAL):  ' FREQH2/
     '    COMBINED TRIPPING FREQUENCY PER OBSERVATION:    ' ADDTRIP/
     '    LONGEST SEQUENCE OF FAILURE DECISIONS:          ' FAILSEQ/
     '    DATA POINT AT WHICH SEQUENCE FIRST OCCURRRED:   ' FSEQPT/
     '    PERCENT TIME A FAILURE MODE SELECTED:           ' ALPH1//
```

```
  triph1 = 0;triph2 = 0;c... = 0; fail1 = 0; fail2 = 0;f... 1pt = 0; f...2pt = 0;
  END;
END;
drop cnt totcnt toth1 toth2 triph1 triph2 failsec fsecpt;
drop fail1 fail1pt fail2 fail2pt f1 f1pt f2 f2pt;
proc means mean noprint;
   var sprt1;
   output out=dummy;
run;

*************************************************************************;

data;set signal;
     set end=EOF;
     y=diff3;sfm=sfm3;v=v3;
     do2sprt


%macro fourrec;
  /*  SAS CODE FOR THE SPECTRAL DECOMPOSITION/RECONSTRUCTION      */
  /* This  macro constructs a curve from raw data using the      */
  /* highest &M Fourier modes(calculated from the power spectral */
  /* density). It reads raw data from the file 'krls' in the     */
  /* SAS library assigned to the libref 'datlib'                 */
  /* and places the constructed curve in the file 'datlib        */
  /* recons**'where ** is the reactor variable code number.      */
  /* Both files are SAS datasets.  The macro also performs a     */
  /* Fisher Kappa Whitetest on the periodograms of the original  */
  /* and residual data.                                          */
  /*                 Written by Kristin Boyer                    */

   /*  Using a number of pts that is a power of 2 to speed up the */
   /*  spectra procedure -- N=16384

 proc spectra data=datlib.krls out=tspec coef p s whitetest;
    var &rawdat;
 proc sort data=tspec out=sortspec;      /* Sort by descending psd to get*/
    by descending p_01 s_01;             /* most prominent periodics     */
 data datlib.&dset;
   set datlib.krls;
   min = _N_ - 1;                                  /* Use 6 highest psc vals */
   do i=1 to &M+1;                                /* Access ith Fourier coef*/
     set sortspec point=i;                        /* and freq and reconstr. */
     if( i  = 1) then power=cos_01/2.0;           /* the curve.             */
     else do;
        power=power + cos_01*cos(freq* min)+sin_01*sin(freq*min);
     end;                                          /* Fourier curve = power */
   end;                                            /* residual = noise      */
   min = min*1.0/120.0;
   noise = &rawdat - power;
   keep min power &rawdat   noise;
   output;
   if _N_>16383 then stop;
  proc spectra data= datlib.&dset out=newpsd whitetest p;
        var noise;
  data newpsd;
    set newpsd;
    p_res = p_01;
    keep p_res freq;
  data  datlib.&specset;
     merge newpsd tspec;
     by freq;
     keep freq p_01 p_res;
  %mend fourrec;
```

```
/*************************  ****************************  ***  ****/

%macro addran;

/*      This macro adds Gaussian random noise to the Fourier curve  */
/*      'power' to create the reconstructed curve 'recons' with the */
/*      same mean and variance as the original curve 'rawdat'.      */
/*      The input dataset should come from the library with         */
/*      libref = 'datlib', and the dataset name should be assigned  */
/*      (with a macro) the the reference name 'dset'                */

proc means data=datlib.&dset var noprint;     /*  Get the variance of the */
   var noise;                                 /*  residual function       */
   output out=twork var=sigmasq;
data datlib.&dset;
  i=1;
  set datlib.&dset;
  set twork point=i;                          /*Generate the random data */
  recons = sigmasq**0.5 *rannor(3) + power; /*and add to 'power         */
  keep min &rawdat power noise recons;
proc means data=datlib.&dset  mean var skewness kurtosis min max;
%mend addran;

/*****************************************************************/

%macro normtst(x=x);

/*     This macro tests the normality of the data 'x'                */
/*     passed as a parameter.  The methods used are                  */
/*     the D'Agostino Pearson K^2 test for the 3rd and 4th           */
/*     moments and the Kolmogorov-Smirnov test.                      */
/*      The input dataset should come from the library with         */
/*      libref = 'datlib', and the dataset name should be assigned  */
/*      (with a macro) the the reference name 'dset'                */
/*               Written by Kristin Hoyer

data twork;
   set datlib.&dset;
proc means data=twork n nmiss skewness kurtosis std noprint;
   var &x;
   output out=stats n=n  skewness=sk kurtosis=k std=sigma;
   /* Compute the skew contribution to the test statistic   */
   /* according to the Johnson Su approximation             */
 data skew;
   set stats;
   y = sk*((n+1)*(n+3)/(6.0*(n-2)))**0.5;
   beta2 = 3*(n*n+27*n-70)*(n+1)*(n+3)/((n-2)*(n+5)*(n+7)*(n+9));
   wsqr = -1.0 + (2.0*(beta2 - 1.0))**0.5;
   delta = 1.0/(0.5*log(wsqr))**0.5;
   alpha = (2.0/(wsqr - 1.0))**0.5;
   z = delta*log(y/alpha + (y*y/(alpha*alpha) + 1.0)**0.5);
   xsqrbl = z**2;
   keep xsqrbl sk;
   output;
   /*  Compute the kurtosis contribution to the test statistic */
   /*  according to the Anscombe and Glynne Approximation      */
  data kurt;
   set stats;
   k = (k+3.0*(n-1)**2/((n-2)*(n-3)));
   b2bar = 3.0*(n-1)/(n+1);
   varb2 = 24.0*n*(n-2)*(n-3)/((n+1)**2 *(n+3)*(n+5));
   x = (k - b2bar)/varb2**0.5;
   beta = 6.0*(n*n-5*n+2)/((n+7)*(n+9))*(6.0*(n+3)*(n+5)
          /(n*(n-2)*(n-3)))**0.5;
   a = 6.0+8.0/beta*(2.0/beta + (1.0+4.0/ beta**2   )**0.5);
   dum =  (1.0-2.0/a)/(1.0+x*(2.0/(a-4.0))**0.5);
```

```
 dum3= (abs(dum))**0.3**  3;
 if (dum < 0.0) then d.  = -dum3;
   z = 1.0 - 2.0/(9.0*a) - dum3;
   z =  z /(2.0/(9.0*a))**0.5;
   xsqrb2 = z**2;
   keep xsqrb2 k;
   output;
/* Get test statistic.  It is Chi square with 2 degrees of freedom */
data _null_;
   file print;
   merge skew kurt;
   ksqr = xsqrb1 + xsqrb2;
   alpha = probchi(ksqr,2,1);
   put 'D Agostino Pearson results for ' "&x"//
   'Skew =       ' sk//
   'Kurtosis=    ' k//
   'Test value   ' ksqr//
   'Alpha =      ' alpha;
proc univariate data=datlib.&dset normal;
   var &x;
%mend normtst;

/*****************************************************************/

%macro runs(r=r);

 /*  This macro performs a 'runs' test on the variable r to determine */
 /*  autocorrelation.  This method was taken from "Checking for       */
 /*  Autocorrelation in Regression Residuals" by Dickey and Brocklebank*/
 /*  Autocorrelation in Regression Residuals" by Dickey and Brocklebank*/
 /*     The input dataset should come from the library with           */
 /*     libref = 'datlib', and the dataset name should be assigned    */
 /*     (with a macro) the the reference name 'dsat'                  */
 /*                    Written by Kristin Boyer

data _null;
   file print;
   set end=eof;                          /* Get negative and positive runs  */
   if &r=. then goto miss;               /* and missing values.             */
   pr=&r>=0;
   np=pr;
   nn=(1-pr);
   if pr ne lag(pr) then runs = 1;
   miss: if eof then do;
     put '                        Run of Signs Test'//
     'Number of Runs '                    runs//
     'Number of Negative Residuals '        nn//
     'Number of Positive Residuals '        np;
 /* Calculate the Z Statistic          */
     u=2*nn*np/(nn+np)+1;
     s=sqrt(2*nn*np*(2*nn*np-nn-np)/((nn+np-1)*(nn+np)**2));
     zp=(runs-u+.5)/s;pp=probnorm(zp);
     zn=(runs-u-.5)/s;pn=1-probnorm(zn);
     put 'Test for positive autocorrelation '/
     'Z= ' zp 'Prob < Z= ' pp//
     'Test for negative autocorrelation'/
     'Z= ' zn 'Prob > Z= 'pn;
   end;
 %mend runs;


 %include macros;
 %let dset=recons14;
 %let rawdat=c14;
 %let specset=psd14;
 %let &M = 10;
 %fourrec
```

```
%addran
%normtst(x = &rawdat)
%normtst(x=noise)
proc means data=data.&dset mean noprint;
var &rawdat;                                    /*  Get the residuals needed */
output out=twork mean=mu;                       /*  for the 'sign' macro     */
data;
    set data.&dset;
    i=1;
    set twork point=i;
    r= &rawdat - mu;
    res=noise;
    keep r res;
    output;
%runs(x=r)
%runs(x=res)
```

APPENDIX B

Listing of the SPRTEXPT.S File

```
;*****************************************************************************;
;                                                                             ;
;                            SPRT_Expert Procedure                            ;
;                                                                             ;
;*****************************************************************************;
;;;
;;;  SPRT_Expert Procedure
(define (SPRT_Expert)

  ;;  RetrieveData Procedure - This internal procedure retrieves the SPRT data
  ;;  from the specified DataFile.  The procedure reads the next NLines lines
  ;;  from the file.  The procedure is recursive, it decrements NLines until
  ;;  NLines reaches 1.  If the end-of-file is reached during the data re-
  ;;  trieval, the eof character is added to the end of the list returned by
  ;;  the procedure.  The procedure returns the NLines read as a list.
  (define RetrieveData
    (lambda (NLines DataFile)
      (let ((CurrentLine (read-line DataFile)))  ; Read next line in DataFile
        (if (eof-object? CurrentLine)            ; If eof reached, return eof
            CurrentLine                          ; character.
            (if (eq? 1 NLines)                   ; If Nlines read, return null.
                (list (Chars-to-List (string- list CurrentLine)))
                (cons (Chars-to-List (string- list CurrentLine))
                      (RetrieveData (- NLines 1) DataFile) ) ) ) ) ) )


  ;;  Chars-to-List Procedure - This internal procedure recursively converts a
  ;;  list of characters that contain embedded 0 and 1 characters into a list
  ;;  of 0 s and 1 s.  Any other character than a 0 or 1 is ignored.
  (define Chars-to-List
    (lambda (Chars)
      (cond ((null? Chars)  ())                  ; If end of list, return null.
            ((char=? (car Chars) #\0)            ; Add 0 to list.
             (cons  0 (Chars-to-List (cdr Chars))) )
            ((char=? (car Chars) #\1)            ; Add 1 to list.
             (cons  1 (Chars-to-List (cdr Chars))) )
            (else (Chars-to-List (cdr Chars))) ) ) )


  ;;  Eof-List? Procedure - This internal procedure searches a list element by
  ;;  element for the eof character.  It returns true if the end-of-file char-
  ;;  acter is found, or false if the list does not contain the eof character.
  (define Eof-List?
    (lambda (Lst)
      (if (eof-object? Lst)                  ; If the current object is the eof
          #t                                 ; character, then return true.
          (if (null? Lst)                    ; If the list is null,
              ()                             ; return false.
              (Eof-List? (cdr Lst)) ) ) ) )


  ;;  ListTrippedSPRTs Procedure - This internal procedure processes the list
  ;;  of SPRT data for a pass through the main do loop.  Each element of the
  ;;  list contains the SPRT data for an independent group of SPRTs.  Each
  ;;  element of the list is processed by the internal FindSPRTs procedure,
  ;;  which returns a list of numbers where each number reprsents the SPRTs
  ;;  that have tripped [i.e., equal to 1].  Thus if the following list of
```

```
;;  SPRTs is passed to FindSPRTs: [0 1 0 0 0 1 0 1], it would return the
;;  following list: [2 6 8].
(define ListTrippedSPRTs
  ;;  Main body of ListTrippedSPRTs
  ;;  This protion of the procedure passes each element of SPRT_List to
  ;;  FindSPRTs and returns a list of the results from FindSPRTs.
  (lambda (SPRT_List)
    (if (null? SPRT_List)
        ()
       (cons (FindSPRTs (car SPRT_List) 1)
             (ListTrippedSPRTs (cdr SPRT_List)) ) ) ) )

;;  FindSPRTs internal procedure:
(define FindSPRTs
  (lambda (Lst I)
    (cond ((null? Lst)  ())          ; Return null if end of Lst is reached.
          ((eq? (car Lst) 1)         ; If SPRT is tripped,
           (cons I                   ; add number to output list.
                 (FindSPRTs (cdr Lst) (+ I 1)) ) )
          (else                      ; If SPRT has not tripped, ignore.
            (FindSPRTs (cdr Lst) (+ I 1)) ) ) ) )


;;  DisplayGroup internal procedure:
;;  This procedure displays the input data for each SPRT group.
(define DisplayGroup
  (lambda (GroupData Count)
    (if (null? (car GroupData))
        ()                           ; Return null at end of GroupData list.
       (begin
         (newline) (display    SPRT Group # ) (display Count) (newline)
         (display    contains  ) (display (caar GroupData)) (display    )
         (display (caddar GroupData))
         (if (eq? (caar GroupData) 1)
             (display   industrial device )
             (display   industrial devices ) )
         (newline) (display    with  ) (display (cadar GroupData))
         (display    ) (display (car (cdddar GroupData)))
         (if (eq? (cadar GroupData) 1)
             (display   redundant sensor. )
             (display   redundant sensors. ) )
         (newline)
         (DisplayGroup (cdr GroupData) (+ 1 Count)) ) ) ) )


;;  Main body of the SPRT_Expert Procedure.
;;  First, intialize local variables.
(let ( (NSPRT_Groups 0) (NDeviceGroups 0) (NVariables 0)
       (NDevices 0) (NSensors 0) (Count 0)
       (DName    ) (VName    )
       (InputFile    ) (InputPort    )
       (GroupData  ()) (SPRT_Data  ()) )

  ;;  Print program title.
  (newline) (newline) (newline)
  (display            SPRT Expert System Simulation Program )
  (newline) (newline) (newline)
  ;;  Query user for number of device groups to be analyzed.
  (display  Enter the number of device groups -   )
  (set! NDeviceGroups (string- number (read-line)  e  d))

  ;;  Start do loop to input data for each device group.
```

```
(do ( (I 1 (+ I 1)) )
    ( (  NDeviceGroups I) #t )
    ;;  Body of do loop.
    (newline) (newline)
    (display  ------------------------------------------------------------ )
    (newline)
    ;;  Query user for device group names.
    (display      DEVICE NAME: ) (newline)
    (display      Enter the name of device group number  ) (display I)
    (display   -   )
    (set! DName (read-line))
    (newline)
    ;;  Query user for the number of independent devices in the group.
    (display      DEVICE NUMBER: ) (newline)
    (display      Enter the number of devices in the  ) (display DName)
    (newline) (display      device group -   )
    (set! NDevices (string- number (read-line)  e  d))
    (newline)
    ;;  Query user for number of physical variables in device group.
    (display      PHYSICAL VARIABLE NUMBER: ) (newline)
    (display      Enter the number of physical variables in the  )
    (display DName) (newline) (display      device group -   )
    (set! NVariables (string- number (read-line)  e  d))

    ;;  Start do loop to input data for each device/physical variable
    ;;  group.
    (do ( (J 1 (+ J 1)) )
    ( (  NVariables J) #t )
        ;;  Body of do loop.
        (newline)
        ;;  Query user for physical variable names.
        (display        PHYSICAL VARIABLE NAME: ) (newline)
        (display        Enter the name of physical variable number  )
        (display J) (newline)
        (display        of the  ) (display DName)
        (display   device group -   )
        (set! VName (read-line))
        (newline)
        ;;  Query user for the number of redundant sensors in the group.
        (display          SENSOR NUMBER: ) (newline)
        (display          Enter the number of redundant sensors for the  )
        (display VName) (newline)
        (display          physical variable in the  ) (display DName)
        (display   device group -   )
        (set! NSensors (string- number (read-line)  e  d))

        ;;  Increment the number of SPRT groups.
        (set! NSPRT_Groups (+ NSPRT_Groups 1))

        ;;  Add SPRT group input data to the GroupData list.
        (set! GroupData (cons
                          (list NDevices NSensors DName VName)
                          GroupData)) ) )

;;  Reverse the order of the GroupData list so that it will be consistent
;;  with the SPRT data.
(set! GroupData (reverse GroupData))

;;  Display SPRT network data.
(newline) (newline)
(display  +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ )
(newline) (newline)
```

```
(display  The number of SPRT Groups in the simulation is  )
(display NSPRT_Groups) (display  . ) (newline)
;;  Call DisplayGroup to display the input variables for each SPRT group.
(DisplayGroup GroupData 1)
(newline) (newline)
(display  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ )
(newline) (newline)

;;  Query user for the name of the data file that contains the SPRT data.
(display  Enter filename for SPRT data -  )
(set! InputFile (read-line))
;;  Open SPRT data file.
(set! InputPort (open-input-file InputFile))

;;  Enter main do loop, which retrieves all of the SPRT data written at
;;  a timestep.  For each timestep of the SPRT program, a number of lines
;;  of data are written to the data file.  Each line contains the SPRT
;;  data (0 s and 1 s) for an independent group of SPRTs.  The number of
;;  lines written during each timestep thus equals NSPRT_Groups.
(do ( (InputData (RetrieveData NSPRT_Groups InputPort)
                 (RetrieveData NSPRT_Groups InputPort)) )
    ( (Eof-List? InputData) (newline) (newline)
      (display  End of  ) (display InputFile)
      (display   reached -- SPRT_Expert terminates. ) (newline) )

    ;;  Main body of the do loop.
    ;;  Update count, which counts the sets of SPRT data written to the
    ;;  file.
    (set! count (+ count 1))
    ;;  Convert the SPRT data for this pass through the loop to a list of
    ;;  tripped SPRTs (i.e., SPRT = 1).
    (set! SPRT_Data (ListTrippedSPRTs InputData))
    (newline)
    (display  ------------------------------------------------------------ )
    (newline) (newline)
    (display    Analyzing SPRT data set number  ) (display count)
    (newline) (newline)

    ;;  Call the SPRT data analysis procedure.  This procedure analyzes
    ;;  the list of tripped SPRTs.  Each element in the list consists of a
    ;;  list of tripped SPRTs.  There are NSPRT_Groups elements in the
    ;;  list - one for each corresponding group of independent SPRTs.
    (Analyze GroupData SPRT_Data 1)

    ;;  Pause the screen so that the user can read the results for the
    ;;  current block of data.
    (display      Hit the enter key to analyze the next  )
    (display  block of data     ) (newline) (read-line) (newline) ) ) )

;;  End of SPRT_Expert Procedure
;;;
```

```
;*****************************************************************************;
;                                                                             ;
;                              Analyze Procedure                              ;
;                                                                             ;
;*****************************************************************************;

;;;
;;;  Analyze Procedure - This procedure contains the logic rules for the
;;;  SPRT Expert System Simulation program.
;;;
(define (Analyze GroupList SPRT_List GroupCount)


  ;;  AnalyzeGroup procedure - This internal procedure controls the flow of
  ;;  SPRT data analysis.  It is passed the input data and the list of tripped
  ;;  SPRTs for a group of independent SPRTs.  Depending upon the number of
  ;;  devices in the group, AnalyzeGroup calls a procedure which performs the
  ;;  actual analysis of the data.
  (define AnalyzeGroup
    (lambda (Inputs SPRTs Count)
      (display  SPRT Group # ) (display Count) (display  : ) (newline)
      (cond ((eq? (car Inputs) 1)              ; The number of devices is 1.
             (if (null? SPRTs)                 ; If no SPRTs have tripped,
                 (begin
                   (display  No SPRTs have tripped. ) (newline) (newline)
                   () )                      ; return null.
                 (SingleDevice (cdr Inputs) SPRTs) ) )
            ((eq? (car Inputs) 2)              ; The number of devices is 2.
             (if (null? SPRTs)                 ; If no SPRTs have tripped,
                 (begin
                   (display  No SPRTs have tripped. ) (newline) (newline)
                   () )                      ; return null.
                 (DualDevice (cdr Inputs) SPRTs) ) )
            (else                              ; The number of devices  = 3.
              (if (null? SPRTs)                ; If no SPRTs have tripped,
                  (begin
                    (display  No SPRTs have tripped. ) (newline) (newline)
                    () )                       ; return null.
                  (MultipleDevice Inputs SPRTs) ) ) ) ) )


  ;;  SingleDevice procedure - This procedure contains the logic for the
  ;;  analysis of a SPRT group which has 1 device.
  (define SingleDevice
    (lambda (InputData SPRT_Data)
      ;;  First use temporary variables to extract the data from the InputData
      ;;  list and the number of tripped SPRTs.
      (let ( (NSensors (car InputData))
             (DName (cadr InputData))
             (VName (caddr InputData))
             (NSPRTs (length SPRT_Data)) )
        ;;  The logic depends upon the number of tripped SPRTs.
        (cond (( = NSPRTs 3)
               ;;  In this case, 3 or more SPRTs have tripped.  First write
               ;;  message showing current device and physical variable.
               (display  For the  ) (display VName)
               (display   physical variable of the  ) (display DName)
               (display   device: ) (newline)
               (if (eq? NSPRTs NSensors)     ; If all of the sensors failed,
                   (begin                    ; write this message.
                     (display  All  ) (display NSPRTs)
                     (display   SPRTs have tripped ) )
                   (begin                    ; or, write this message.
```

```
        (display  These  ) (display NSPRTs) (display   of the  )
        (display NSensors) (display   SPRTs have tripped -   )
        (newline)
        (DisplaySPRTs 1 NSensors SPRT_Data) ) )
  ;;  We conclude that the device has failed.
  (newline) (display  *** THE  ) (display DName)
  (display   DEVICE HAS FAILED *** ) (newline)
  (newline) )

 ((eq? NSPRTs 2)
  ;;  In this case, 2 SPRTs have tripped.  First write
  ;;  message showing current device and physical variable.
  (display  For the  ) (display VName)
  (display   physical variable of the  ) (display DName)
  (display   device: ) (newline)
  ;;  List the tripped SPRTs.
  (display  These 2 of the  ) (display NSensors)
  (display   SPRTs have tripped -  ) (newline)
  (DisplaySPRTs 1 NSensors SPRT_Data) (newline)
  ;;  Next, determine if a sensor or the device has failed.
  (cond ((eq? (car SPRT_Data) (- (cadr SPRT_Data) 1))
         ;;  We can conclude that a sensor has failed, because
         ;;  the tripped SPRTs are next to each other.
         (display  *** SENSOR NUMBER A )
         (display (cadr SPRT_Data)) (display   HAS FAILED *** )
         (newline) (newline))
        ((and (eq? (car SPRT_Data) 1)
              (eq? (cadr SPRT_Data) NSensors))
         ;;  We can conclude that the first sensor has failed,
         ;;  because the first and last SPRTs have tripped.
         (display  *** SENSOR NUMBER A1 HAS FAILED *** )
         (newline) (newline))
        (else
          ;;  We can conclude that the device has failed, since
          ;;  the tripped SPRTs are not adjacent.
          (display  *** THE  ) (display DName)
          (display   DEVICE HAS FAILED *** ) (newline)
          (newline) ) ) )

 ((eq? NSPRTs 1)
  ;;  In this case, 1 SPRT has tripped.  This is due either to
  ;;  a early indication of a sensor or device failure, or to
  ;;  a spurious SPRT trip.  We conclude a possible failure.
  (display  For the  ) (display VName)
  (display   physical variable of the  ) (display DName)
  (display   device: ) (newline)
  (display  One SPRT has tripped -   )
  (DisplaySPRTs 1 NSensors SPRT_Data) (newline)
  (display  *** THE  ) (display DName) (display   DEVICE,  )
  (if (eq? (car SPRT_Data) NSensors)
      (begin
        (display  SENSOR NUMBER A1, ) (newline)
        (display       OR SENSOR NUMBER A )
        (display NSensors) (display   MAY BE FAILING *** )
        (newline) (newline) )
      (begin
        (display  SENSOR NUMBER A ) (display (car SPRT_Data))
        (display  , ) (newline)
        (display       OR SENSOR NUMBER A )
        (display (+ (car SPRT_Data) 1))
        (display   MAY BE FAILING *** )
        (newline) (newline) ) ) ) ) ) )
```

```
;;  DualDevice procedure - This procedure contains the logic for the
;;  analysis of a SPRT group which has 2 devices.
(define DualDevice
  (lambda (InputData SPRT_Data)
    ;;  First use temporary variables to extract the data from the InputData
    ;;  list and the number of tripped SPRTs.
    (let ( (NSensors (car InputData))
           (DName (cadr InputData))
           (VName (caddr InputData))
           (NSPRTs (length SPRT_Data)) )
      ;;  The logic depends upon the number of tripped SPRTs.
      (cond (( = NSPRTs 3)
             ;;  In this case, 3 or more SPRTs have tripped.  First write
             ;;  message showing current device and physical variable.
             (display  For the  ) (display VName)
             (display   physical variable of the  ) (display DName)
             (display   devices: ) (newline)
             (if (eq? NSPRTs (* NSensors 2)) ; If all of the sensors failed,
                 (begin                      ; write this message,
                   (display  All  ) (display NSPRTs)
                   (display   SPRTs have tripped ) )
                 (begin                      ; or, write this message.
                   (display  These  ) (display NSPRTs) (display   of the  )
                   (display (* NSensors 2))
                   (display   SPRTs have tripped -   ) (newline)
                   (DisplaySPRTs 2 NSensors SPRT_Data) ) )
             ;;  We conclude that the device has failed.
             (newline) (display  *** ONE OR BOTH OF THE  ) (display DName)
             (display   DEVICES HAVE FAILED *** ) (newline)
             (newline) )

            ((eq? NSPRTs 2)
             ;;  In this case, 2 SPRTs have tripped.  First write
             ;;  message showing current device and physical variable.
             (display  For the  ) (display VName)
             (display   physical variable of the  ) (display DName)
             (display   devices: ) (newline)
             ;;  List the tripped SPRTs.
             (display  These 2 of the  ) (display (* NSensors 2))
             (display   SPRTs have tripped -  ) (newline)
             (DisplaySPRTs 2 NSensors SPRT_Data) (newline)
             ;;  Next, determine if a sensor or the device has failed.
             (cond ((eq? (car SPRT_Data) (- (cadr SPRT_Data) 1))
                    ;;  We can conclude that a sensor has failed, because
                    ;;  the tripped SPRTs are next to each other.
                    (display  *** SENSOR NUMBER  )
                    (if (even? (car SPRT_Data))
                        (begin
                          (display  B )
                          (display (+ (/ (car SPRT_Data) 2) 1)) )
                        (begin
                          (display  A )
                          (display (+ (quotient (car SPRT_Data) 2) 1)) ) )
                    (display   HAS FAILED *** )
                    (newline) (newline))
                   ((and (eq? (car SPRT_Data) 1)
                         (eq? (cadr SPRT_Data) (* NSensors 2)))
                    ;;  We can conclude that the first sensor of the second
                    ;;  device has failed, because the first and last SPRTs
                    ;;  have tripped.
                    (display  *** SENSOR NUMBER B1 HAS FAILED *** )
```

```
                (newline) (newline))
               (else
                 ;;  We can conclude that the device has failed, since
                 ;;  the tripped SPRTs are not adjacent.
                 (display  *** ONE OR BOTH OF THE  )
                 (display DName) (display   DEVICES HAVE FAILED *** )
                 (newline) (newline) ) ) )

          ((eq? NSPRTs 1)
           ;;  In this case, 1 SPRT has tripped.  This is due either to
           ;;  a early indication of a sensor or device failure, or to
           ;;  a spurious SPRT trip.  We conclude a possible failure.
           (display  For the  ) (display VName)
           (display   physical variable of the  ) (display DName)
           (display   devices: ) (newline)
           (display  One SPRT has tripped -   )
           (DisplaySPRTs 2 NSensors SPRT_Data) (newline)
           (display  *** ONE OR BOTH OF THE  ) (display DName)
           (display   DEVICES, ) (newline)
           (if (eq? (car SPRT_Data) (* NSensors 2))
               (begin
                 (display      SENSOR NUMBER A ) (display NSensors)
                 (display  , OR SENSOR NUMBER B1 MAY BE FAILING *** )
                 (newline) (newline) )
               (begin
                 (display      SENSOR NUMBER A )
                 (if (even? (car SPRT_Data))
                     (begin
                       (display (/ (car SPRT_Data) 2))
                       (display   OR SENSOR NUMBER B )
                       (display (+ (/ (car SPRT_Data) 2) 1))
                       (display   MAY BE FAILING *** ) )
                     (begin
                       (display (+ (quotient (car SPRT_Data) 2) 1))
                       (display   OR SENSOR NUMBER B )
                       (display (+ (quotient (car SPRT_Data) 2) 1))
                       (display   MAY BE FAILING *** ) ) )
                 (newline) (newline) ) ) ) ) ) ) )


;;  MultipleDevice procedure - This procedure contains the logic for the
;;  analysis of a SPRT group which has 3 or more devices.
(define MultipleDevice
  (lambda (InputData SPRT_Data)
    ;;  First use temporary variables to extract the data from the InputData
    ;;  list, the number of tripped SPRTs, the names of the tripped SPRTs,
    ;;  and the list of tripped SPRTs for the last device, and two temporary
    ;;  variables for the main do loop in the procedure.
    (let* ( (NDevices (car InputData))
            (NSensors (cadr InputData))
            (DName (caddr InputData))
            (VName (cadddr InputData))
            (NSPRTs (length SPRT_Data))
            (DeviceList (NameTrippedSPRTs NDevices SPRT_Data))
            (PreviousDevice (car (reverse DeviceList)))
            (CurrentDevice  ())
            (NextDevice  ()) )
      ;;  The logic depends upon the number of tripped SPRTs.
      (cond ((eq? NSPRTs (* NSensors NDevices))
             ;;  In this case, all of the SPRTs have tripped.  First write
             ;;  a message showing current the device and physical variable.
             (display  For the  ) (display VName)
```

```
(display   physical variable of the  ) (display DName)
(display   devices: ) (newline)
(display  All  ) (display NSPRTs)
(display   SPRTs have tripped ) (newline)
(display  *** ALL  ) (display NDevices) (display   OF THE  )
(display DName) (display   DEVICES HAVE FAILED *** )
(newline) (newline) )

((  NSPRTs 0)
 ;;  In this case, 1 or more of the SPRTs have tripped.  So we
 ;;  determine if any device or sensors have failed.  First
 ;;  write a message showing the current device and physical
;;  variable.  Then the names of the tripped SPRTs are displayed.
 (display  For the  ) (display VName)
 (display   physical variable of the  ) (display DName)
 (display   devices: ) (newline)
 (display  These  ) (display NSPRTs) (display   of the  )
 (display (* NSensors NDevices))
 (display   SPRTs have tripped -   ) (newline)
 (DisplaySPRTs NDevices NSensors SPRT_Data) (newline)
 ;;  A do loop is used to step through the list of names of
 ;;  tripped SPRTs.  For each device, we can determine if the
 ;;  device has failed, may be failing, or if one of its sensors
 ;;  has or may be failing.  The do list steps from device 1 to
 ;;  NDevices.
 (do ( (SubList DeviceList (cdr SubList)) )
     ( (null? SubList) (newline) )
     ;;  Body of do loop.
     ;;  First I reset the temporary variables which contain
     ;;  the list of tripped SPRTs for the current and next
     ;;  devices.
     (set! CurrentDevice (car SubList))
     ;;  If this is the last pass through the do loop, then
     ;;  current device is number NDevices and the next device
     ;;  is the first device [i.e., device number A].
     (if (null? (cdr SubList))
         (set! NextDevice (car DeviceList))
         (set! NextDevice (cadr SubList)) )
     ;;  The logic depends upon the number of tripped SPRTs for
     ;;  the current and next devices.
     (cond ( (and (not (null? CurrentDevice))
                  (not (null? NextDevice)) )
             ;;  In this case, the current device and the next
             ;;  device contain tripped SPRTs.
             ;;  Next we check to see if the tripped SPRTs
             ;;  imply a sensor has failed.
             (if (and (eq? (length CurrentDevice) 1)
                      (eq? (length NextDevice) 1)
                      (eq? (cadar CurrentDevice)
                           (cadar NextDevice)) )
                 ;;  In this case, the current and next devices
                 ;;  contain 1 tripped SPRT and the sensor num-
                 ;;  bers of the SPRTs are the same.
                 (if (eq? NSensors 1)
                     ;;  If the devices contain only 1 sensor,
                     ;;  then either the device or sensor has
                     ;;  failed.
                     (begin
                       (display  *** DEVICE NUMBER  )
                       (display (caar NextDevice))
                       (display   OF THE  ) (display DName)
                       (display   DEVICES. ) (newline)
```

```
                    (display      OR SENSOR NUMBER  )
                    (display (caar NextDevice))
                    (display (cadar CurrentDevice))
                    (display   HAS FAILED *** ) (newline) )
                  ;;  The devices contain more than 1 sensor.
                  ;;  We can conclude a sensor has failed.
                  (begin
                    (display  *** SENSOR NUMBER  )
                    (display (caar NextDevice))
                    (display (cadar CurrentDevice))
                    (display   HAS FAILED *** ) (newline) ) )
                ;;  In this case more than 1 SPRT in either of
                ;;  the devices has tripped, therefore we can
                ;;  conclude that the device has failed.
                (begin
                  (display  *** DEVICE NUMBER  )
                  (display (caar NextDevice))
                  (display   OF THE  ) (display DName)
                  (display   DEVICES HAS FAILED *** )
                  (newline) ) ) )
              ;;  Next we check to see if the current device
              ;;  contains tripped SPRTs while the previous and
              ;;  next devices don t.
              ( (and (  (length CurrentDevice) 0)
                     (eq? (length PreviousDevice) 0)
                     (eq? (length NextDevice) 0) )
                ;;  If only one SPRT has tripped, then we can t
                ;;  conclude a failure, only that failure of the
                ;;  current device or a sensor is possible.
                (if (eq? (length CurrentDevice) 1)
                    (begin
                      (display  *** DEVICE NUMBER  )
                      (display (caar CurrentDevice))
                      (display   OR DEVICE NUMBER  )
                      (display (caddar CurrentDevice))
                      (display   OF THE  ) (display DName)
                      (display   DEVICES, ) (newline)
                      (display      SENSOR NUMBER  )
                      (display (caar CurrentDevice))
                      (display (cadar CurrentDevice))
                      (display  , OR SENSOR NUMBER  )
                      (display (caddar CurrentDevice))
                      (display (cadar CurrentDevice))
                      (display   MAY BE FAILING *** ) (newline) )
                    ;;  In this case, more than 1 SPRT in the
                    ;;  current device has tripped.  Therefore we
                    ;;  can conclude that the current or the next
                    ;;  device has failed.
                    (begin
                      (display  *** DEVICE NUMBER  )
                      (display (caar CurrentDevice))
                      (display   OR DEVICE NUMBER  )
                      (display (caddar CurrentDevice))
                      (display   OF THE  ) (display DName)
                      (newline)
                      (display      DEVICES HAS FAILED *** )
                      (newline) ) ) ) )
            ;;  The last step is to update the PreviousDevice variable.
            (set! PreviousDevice CurrentDevice) ) ) ) ) ) )


;;  NameTrippedSPRTs procedure - This internal procedure takes a list of
```

```
;;  tripped SPRTs and converts each number of the tripped SPRT into the SPRT
;;  name (e.g., A 1 B 1, is the corresponding SPRT name for tripped SPRT 1].
;;  The output is a list of NDevs elements.  Each of the elements contains
;;  the name of tripped SPRTs for the corresponding device number (e.g., the
;;  3rd element contains the names of the tripped SPRTs for device C].  If
;;  none of the SPRTs for a device have tripped, then the corresponding ele-
;;  ment is null.  This procedure is applied only to device groups that
;;  contain 3 or more devices.
(define NameTrippedSPRTs
  (lambda (NDevs SPRT_List)
    ;;  Create local list to store output data.
    (let ( (DevList  ()) )
      ;;  A do loop is used to step through each of the devices, from
      ;;  device A to device NDevs.  The loop variable, IDev, is the current
      ;;  device number.
      (do ( (IDev 1 (+ IDev 1)) )
          ( (  NDevs IDev) (reverse DevList) )  ; Reverse order of output.
          (set! DevList
                ;;  Apply the following unnamed function to each of the
                ;;  elements in SPRT_List.  Any null elements are removed
                ;;  from the list returned by map.  Finally, the returned
                ;;  list is added to DevList.
                (cons
                  (RemoveNull             ; Call function to remove nulls.
                    (map
                      (lambda (SPRT)        ; Unnamed internal function.
                        (if (eq? (modulo SPRT NDevs) (modulo IDev NDevs))
                            ;;  If the current element in SPRT_List is a
                            ;;  multiple of IDev, then:
                            (if (eq? (modulo SPRT NDevs) 0)
                                ;;  In this case the devices in the SPRT are
                                ;;  device A and device NDevs.
                                (list
                                  ;;  Return list containing: letter ref-
                                  ;;  ering to first device in the SPRT,
                                  (ascii- symbol (+ NDevs 64))
                                  ;;  number refering to the sensor,
                                  (quotient SPRT NDevs)
                                  ;;  the letter of the second device,
                                   A
                                  ;;  and the sensor number again.
                                  (quotient SPRT NDevs) )
                                ;;  Else,
                                (list
                                  ;;  Return list containing: letter ref-
                                  ;;  ering to first device in the SPRT,
                                  (ascii- symbol (+ IDev 64))
                                  ;;  number refering to the sensor,
                                  (+ (quotient SPRT NDevs) 1)
                                  ;;  the letter of the second device,
                                  (ascii- symbol (+ IDev 65))
                                  ;;  and the sensor number again.
                                  (+ (quotient SPRT NDevs) 1) ) ) ) )
                      SPRT_List ) )
                  ;;  Add result from RemoveNull to DevList.
                  DevList ) ) ) ) ) )


;;  RemoveNull procedure - This internal procedure recursively removes the
;;  null elements from an input list (Lst).
(define RemoveNull
  (lambda (Lst)
```

```
    (if (null? Lst)                          ; If end of Lst is reached,
        ()                                   ; return null.
        (if (null? (car Lst))                ; If 1st element in Lst is null,
            (RemoveNull (cdr Lst))           ; ignore it.
            (cons (car Lst)                  ; Else add 1st element to output.
                  (RemoveNull (cdr Lst)) ) ) ) ) )


;;  DisplaySPRTs Procedure - This internal procedure outputs the names of
;;  the tripped SPRTs in Lst.  For each tripped SPRT listed in Lst, the pro-
;;  cedure determines the name [e.g., A1-A2, or A1-B1] of the SPRT and
;;  writes it to the screen.
(define DisplaySPRTs
  (lambda (NDevs NSens Lst)
    (cond ((null? Lst) #t)                   ; Lst is empty, don t do anything.
          ((eq? NDevs 1)                     ; Case for single device SPRT.
           (if (eq? (car Lst) NSens)
               (begin
                 ;;  In this case, the last SPRT [AN-A1] has tripped.
                 (display  A ) (display NSens) (display  -A1 )
                 ;;  Display triped SPRTs in the remainder of the list.
                 (DisplaySPRTs NDevs NSens (cdr Lst)) )
               (begin
                 ;;  SPRT number AX-AX+1, where X is the first number in
                 ;;  the list, has tripped.
                 (display  A ) (display (car Lst)) (display  -A )
                 (display (+ (car Lst) 1)) (display    )
                 ;;  Display tripped SPRTs in the remainder of the list.
                 (DisplaySPRTs NDevs NSens (cdr Lst)) ) ) )
          ((eq? NDevs 2)                     ; Case for dual device SPRT.
           (if (eq? (car Lst) (* NSens 2))
               (begin
                 ;;  In this case, the last SPRT [AN-B1] has tripped.
                 (display  A ) (display NSens) (display  -B1 )
                 ;;  Display tripped SPRTs in the remainder of the list.
                 (DisplaySPRTs NDevs NSens (cdr Lst)) )
               (begin
                 (display  A )
                 (if (even? (car Lst))
                     (begin
                       ;;  SPRT number AX-BX+1, where X is the first number
                       ;;  in the list divided by 2, has tripped.
                       (display (/ (car Lst) 2)) (display  -B )
                       (display (+ (/ (car Lst) 2) 1)) )
                     (begin
                       ;;  SPRT number AX-BX, where X is .5 + the first num-
                       ;;  ber in the list divided by 2, has tripped.
                       (display (+ (quotient (car Lst) 2) 1))
                       (display  -B )
                       (display (+ (quotient (car Lst) 2) 1)) ) )
                 (display    )
                 ;;  Display tripped SPRTs in the remainder of the list.
                 (DisplaySPRTs NDevs NSens (cdr Lst)) ) ) )
          (else                              ; Case for multiple device SPRT.
            (if (eq? (modulo (car Lst) NDevs) 0)
                ;;  If the tripped SPRT is a multiple of the number of
                ;;  devices, then the SPRT number is NX-AX, where N is the
                ;;  number of devices, and X is the sensor number.
                (begin
                  ;;  Letter representing device N.
                  (display (ascii- symbol (+ NDevs 64)))
                  ;;  Number of the sensor.
```

```
                (display (quotient (car Lst) NDevs))
                ;;  Letter representing device A.
                (display "-A")
                ;;  Number of the sensor.
                (display (quotient (car Lst) NDevs)) )
              (begin
                ;;  If the tripped SPRT is not a multiple of the number of
                ;;  devices, then the number of the first device is given
                ;;  by the remainder of SPRT/NDevs.
                ;;  Letter representing first device in tripped SPRT.
                (display (ascii->symbol (+ (modulo (car Lst) NDevs) 64)))
                ;;  Number of the sensor.
                (display (+ (quotient (car Lst) NDevs) 1))
                (display "-")
                ;;  Letter representing second device in tripped SPRT.
                (display (ascii->symbol (+ (modulo (car Lst) NDevs) 65)))
                ;;  Number of the sensor.
                (display (+ (quotient (car Lst) NDevs) 1)) ) )
            (display " ")
            ;;  Display tripped SPRTs in the remainder of the list.
            (DisplaySPRTs NDevs NSens (cdr Lst)) ) ) ) )


;;  Main body of Analyze procedure.
;;
;;  The Analyze procedure is passed input data and tripped SPRT data super
;;  lists.  Each element of these lists contains the input data and list of
;;  tripped SPRTs for a corresponding group of independent SPRTs.  This por-
;;  tion of the procedure is a control routine which each element of the
;;  super lists to the AnalyzeGroup procedure.
(if (null? SPRT_List)                ; Quit if end of list is reached.
    #t
    (begin
      (AnalyzeGroup (car GroupList)  ; Analyze 1st element of the lists.
                    (car SPRT_List)
                    GroupCount)
      (Analyze (cdr GroupList)       ; Analyze remainder of the lists.
               (cdr SPRT_List)
               (+ 1 GroupCount)) ) ) )

;;  End of the Analyze procedure.
;;=============================================================================;
```

What Is Claimed Is:

1. A method of testing both an industrial process and a sensor for determining fault conditions therein, comprising the steps of:

determining automatically, using computer means, a configuration of a minimum number of sensor pairs needed to test the industrial process and the sensor for state of operation;

operating at least a first and second sensor to form at least one sensor pair to redundantly detect at least one physical variable of the industrial process to provide a first signal from said first sensor and a second signal from said second sensor, each said signal being characteristic of the one physical variable;

obtaining a difference function characteristic of the arithmetic difference pairwise between said first signal and said second signal at each of a plurality of different times of sensing the one physical variable;

obtaining a frequency domain transformation of said first difference function to procure Fourier coefficients corresponding to Fourier frequencies;

generating a composite function over time domain using the Fourier coefficients;

obtaining a residual function over time by determining the arithmetic difference between the difference function and the composite function, the residual function being substantially free of serially correlated noise;

operating on the residual function using the computer means for performing a statistical analysis technique to determine whether an alarm condition is present in at least one of the industrial process and the sensor, the residual function including white noise characteristics of an uncorrelated function of reduced skewness relative to the difference function and being input to the statistical analysis technique; and

said at least one sensor pair providing alarm information to an operator of the industrial process allowing modification of at least one of the industrial process and said at least first and second sensor when an alarm condition is detected.

2. The method described is Claim 1 wherein said computer means comprises an artificial intelligence system.

3. The method as defined in Claim 1 wherein the residual function comprises reduced Markov dependent noise.

4. The method as defined in Claim 1 wherein the industrial process comprises at least one of a chemical process, a mechanical process and an electrical operational process.

5. The method as defined in Claim 1 wherein the step of obtaining Fourier coefficients comprise iteratively determining the minimum number of Fourier harmonics able to generate the composite function,

6. The method as defined in Claim 1 further including at least one of the steps of modifying the industrial process or changing the sensor responsive to the alarm condition.

7. A method of testing both an industrial process and a sensor for determining fault conditions therein, comprising the steps of:

determining, using computer means, a configuration of a minimum number of sensor pair signals needed to characterize the industrial process and the sensor state of operation;

operating at least a first sensor to detect at least one physical variable of the industrial process to provide a real signal from said first sensor;

generating an artificial signal characteristic of the one physical variable;

forming a sensor pair signal from said real signal and said artificial signal;

obtaining a difference function characteristic of the difference pairwise between said real signal and said artificial signal at each of a plurality of different times of sensing the one physical variable;

obtaining a frequency domain transformation of said difference function;

generating a composite function over a time domain;

obtaining a residual function over time by determining the difference between the frequency transformed difference function and the composite function;

operating on the residual function using the computer means for performing a statistical analysis technique to determine whether an alarm condition is present in at least one of the industrial process and the first sensor, the residual function including white noise characteristics of an uncorrelated signal of reduced skewness relative to the difference function and being input to the statistical analysis technique; and

said first sensor providing alarm information to an operator of the industrial process allowing modification of at least one of the industrial process and the first sensor when an alarm condition is detected.

8. The method as defined in Claim 7 wherein the step of obtaining a frequency domain transformation comprises performing a Fourier transformation.

9. The method as defined in Claim 7 wherein the steps of obtaining a composite function over time comprises performing an auto regressive moving average analysis.

10. The method as defined in Claim 7 further including the step of determining a difference function for both the artificial signal and the real signal, as well as a separate pair of real signals.

11. The method as defined in Claim 7 wherein the residual function comprises reduced Markov dependent noise.

12. The method as defined in Claim 8 wherein the step of obtaining a frequency domain transformation comprises obtaining Fourier coefficients iteratively to determine the minimum number of Fourier harmonics able to generate the composite function.

13. A system for automatically configuring sensors for testing both an industrial process and a sensor for determining a fault condition therein, comprising:

computer means for automatically configuring a minimum number of sensor pair signals needed to characterize the industrial process and the sensor state of operation;

at least a first sensor to detect at least one physical variable of the industrial process to provide a first signal from said first sensor;

first means for generating a second sensor signal for comparison with said first signal from said first sensor;

second means for determining a difference function characteristic of the arithmetic difference pairwise between said first signal and said second signal at each of a plurality of different times of sensing the one physical variable;

third means for obtaining a residual function over time by means for determining the arithmetic difference between the difference function and the composite function, the residual function including white noise characteristics of an uncorrelated signal of reduced skewness;

fourth means for operating on the residual function, said fourth means including the computer means for executing a computer program for performing a statistical analysis technique and for determining whether an alarm condition is present in at least one of the industrial process and the sensor and with said second means, said third means, and said fourth means cooperatively providing a function comprised of said white noise characteristics of uncorrelated signal of reduced skewness relative to the difference function as an input to the statistical analysis technique; and

means for providing information allowing modification of at least one of the industrial process and the sensor when an alarm condition is detected.

14. The system as defined in Claim 13 further including means for obtaining a frequency domain transformation of said difference function.

15. The system as defined in Claim 13 wherein said computer means comprises an artificial intelligence system.

16. The system as defined in Claim 13 wherein said means for generating a second signal comprises the computer means for executing a computer program.

17. The system as defined in Claim 16 wherein the computer program includes an autoregressive moving average procedure.

18. The system as defined in Claim 13 wherein the system includes at least one pair of sensors for detecting each of the physical variables.

19. The system as defined in Claim 13 wherein said computer means executes a computer program including a statistical probability ratio test on the residual function.

20. The system as defined in Claim 13 further including means for changing at least one of the industrial process and substituting another sensor for a defective sensor.

Fig. 1

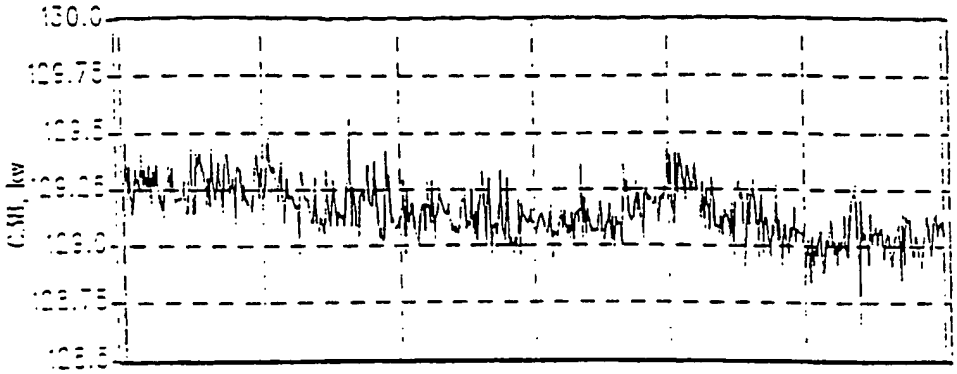


Fig. 2

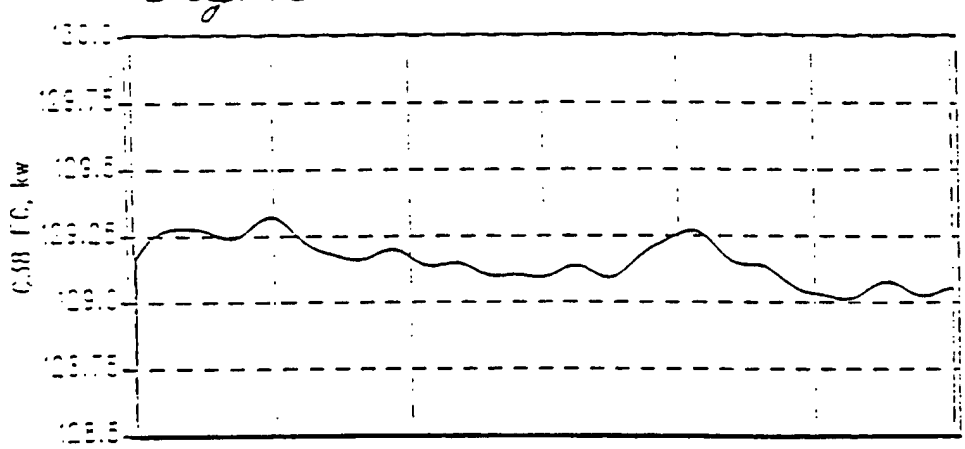
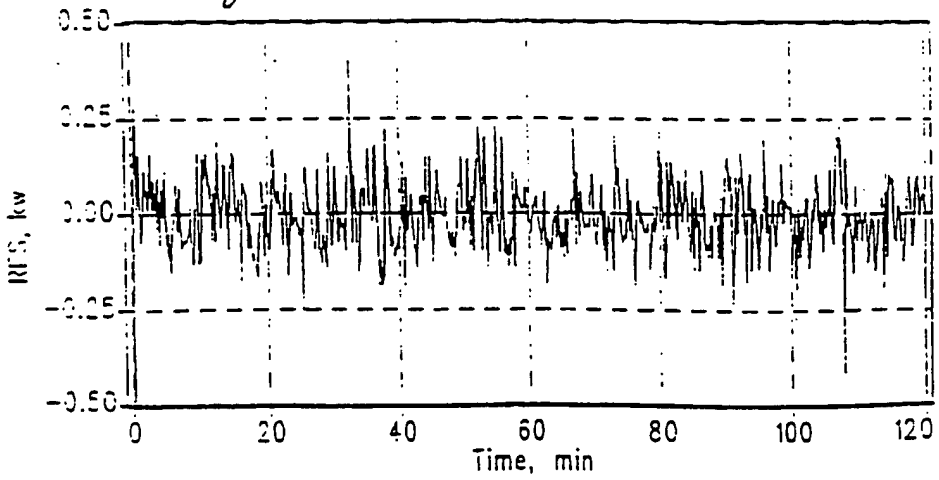


Fig. 3



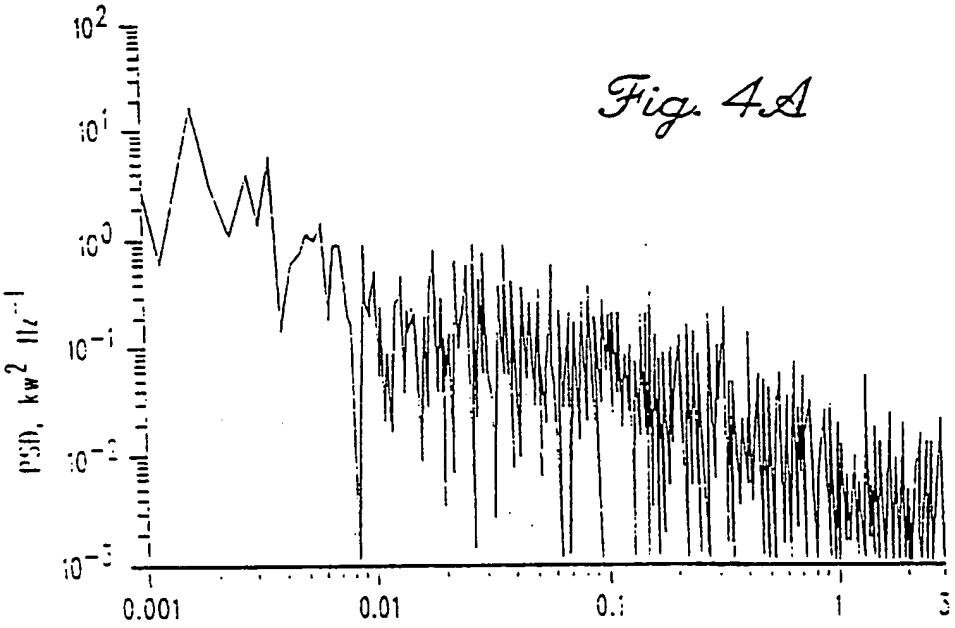


Fig. 4A

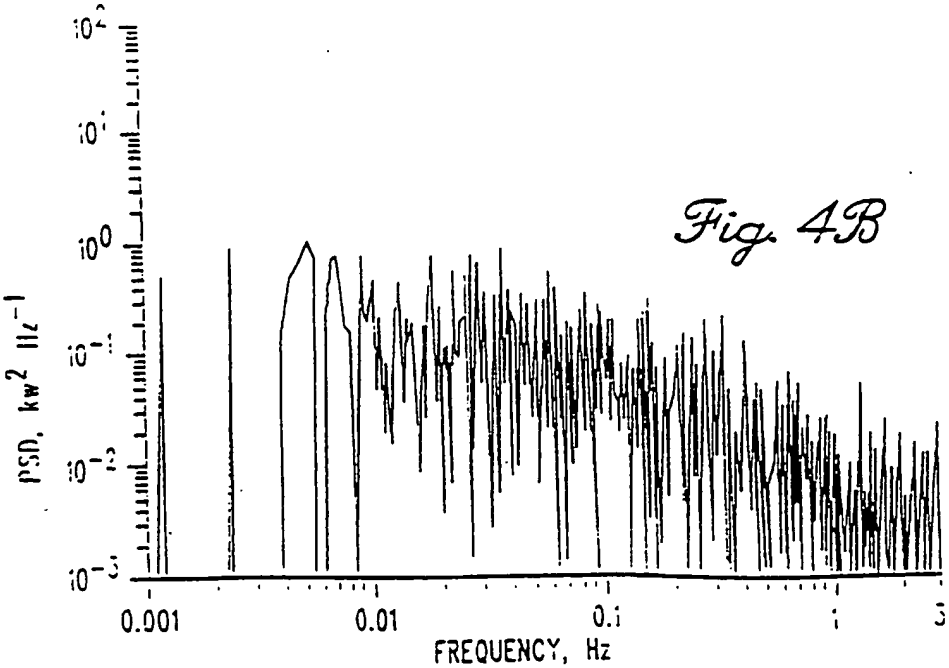


Fig. 4B

Fig. 5A

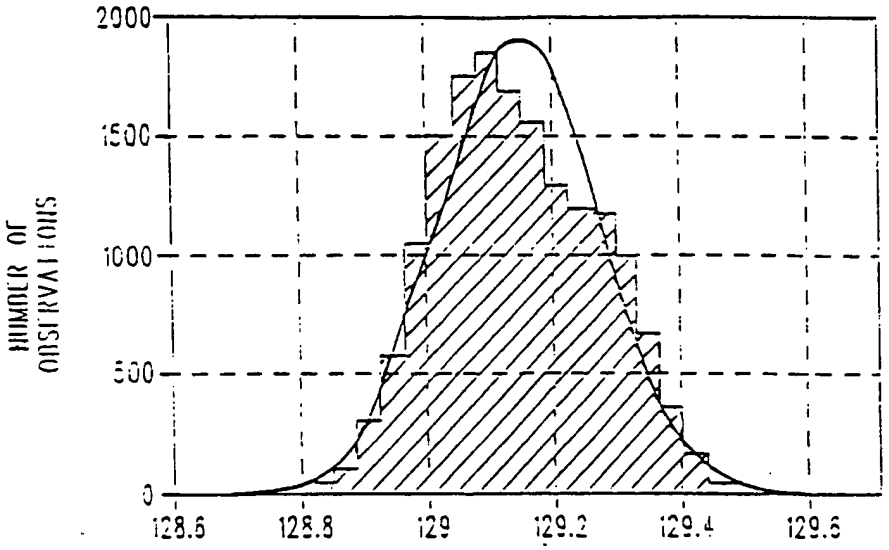

2000
1500
1000
500
0
NUMBER OF OBSERVATIONS
128.6
128.8
129
129.2
129.4
129.6

Fig. 5B

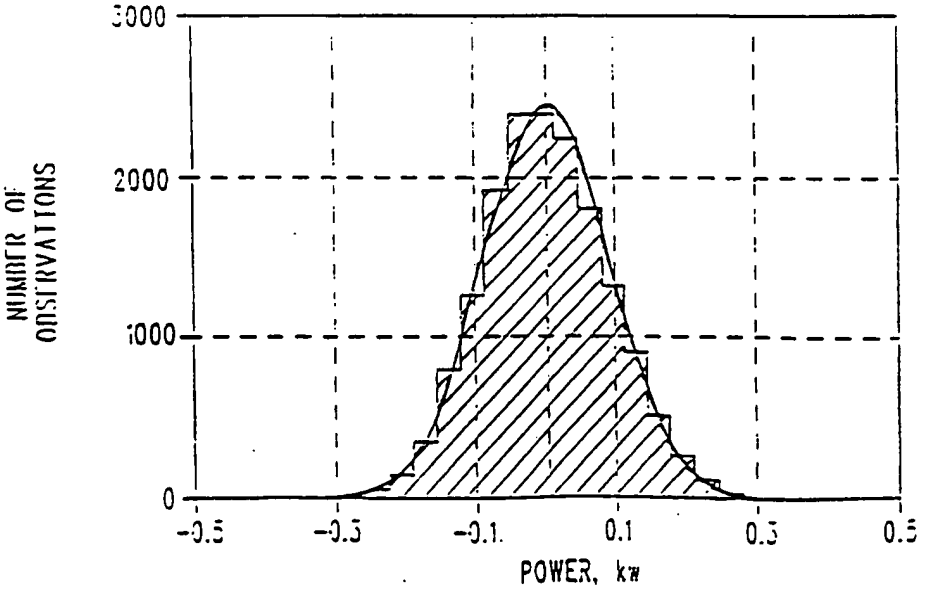

3000
2000
1000
0
NUMBER OF OBSERVATIONS
-0.5
-0.3
-0.1
0.1
0.3
0.5
POWER, kw

Fig. 6A

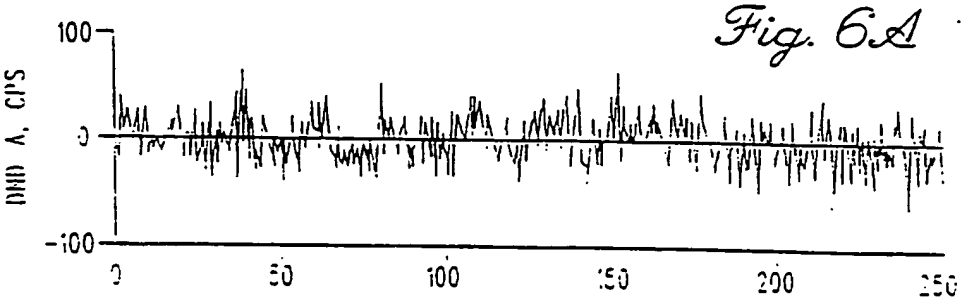


Fig. 6B

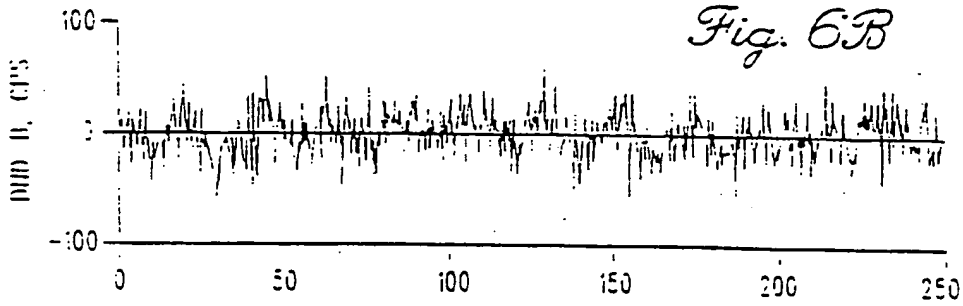


Fig. 6C

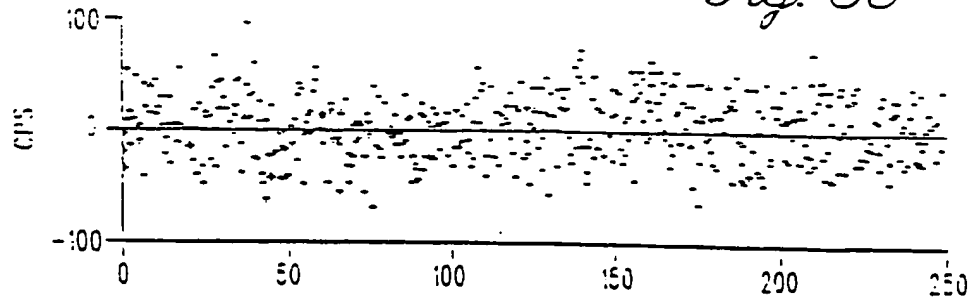


Fig. 6D

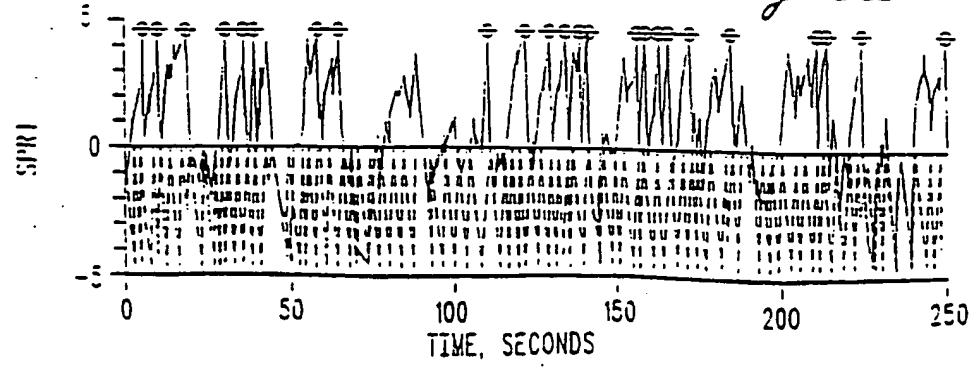


Fig. 7A

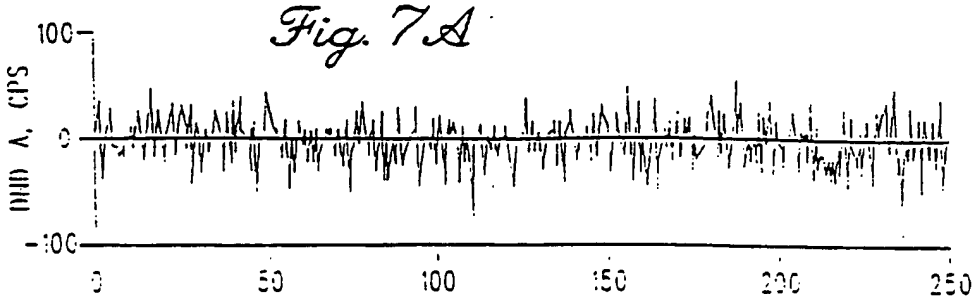


Fig. 7B

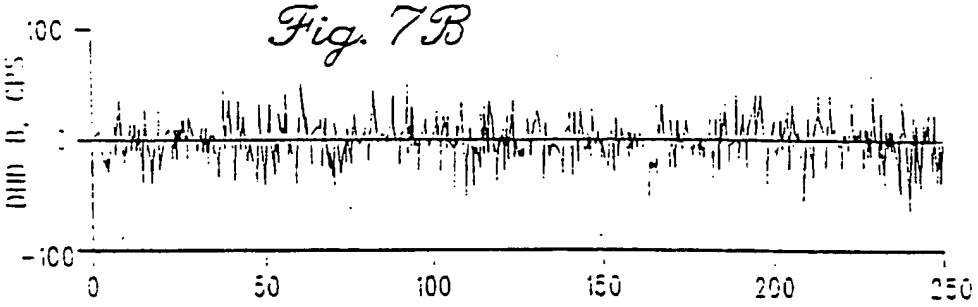


Fig. 7C

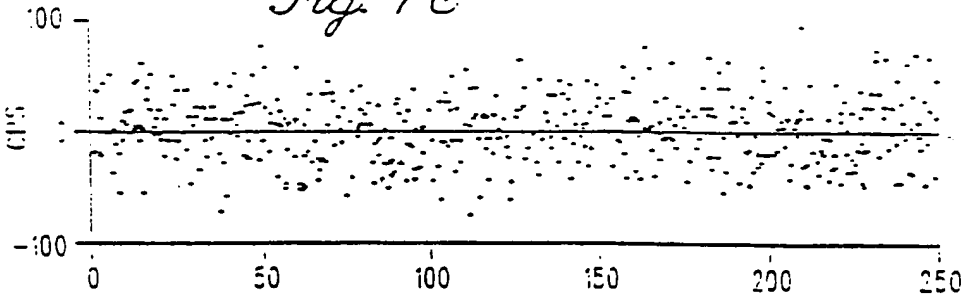


Fig. 7D

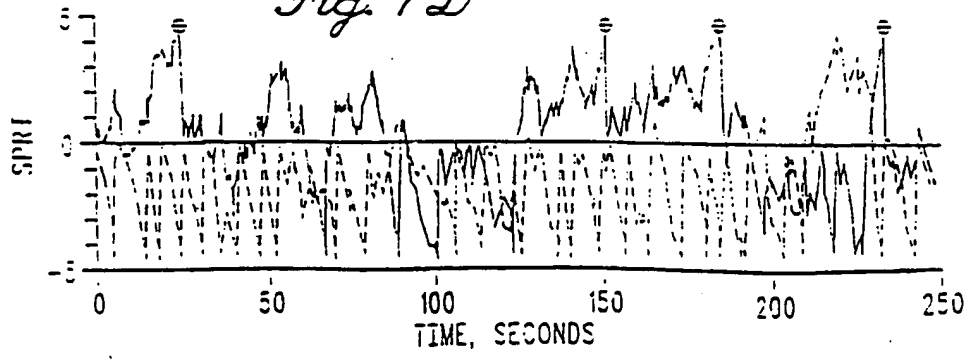


Fig. 8A

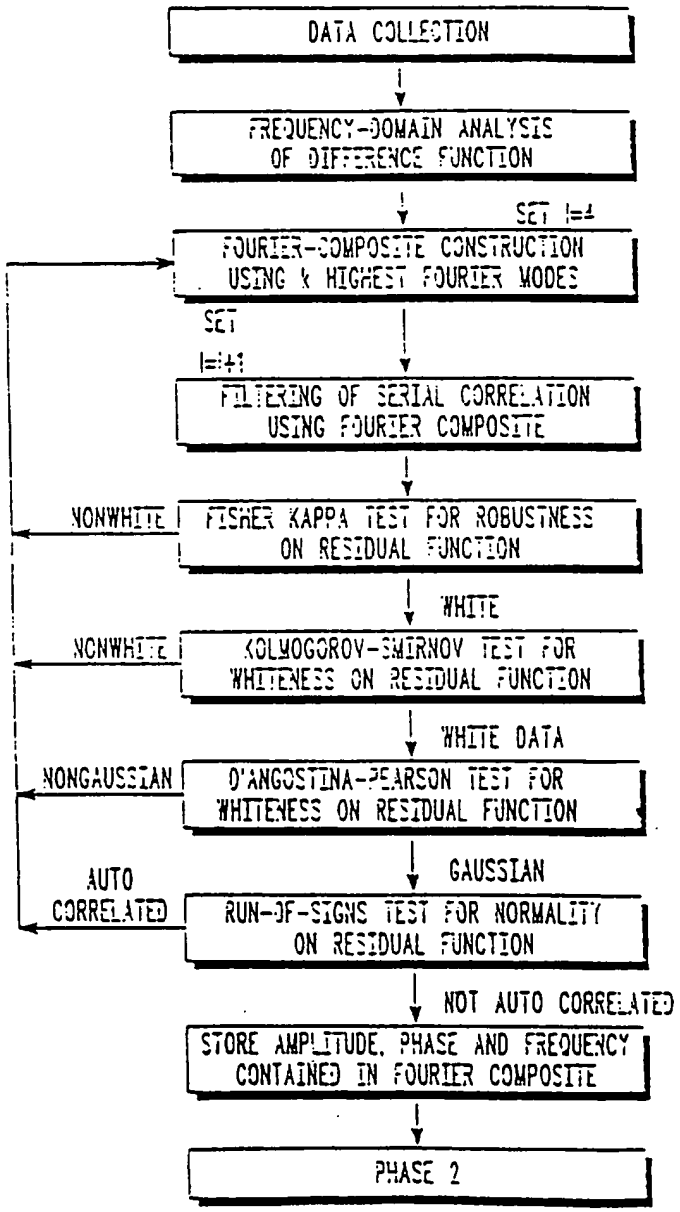
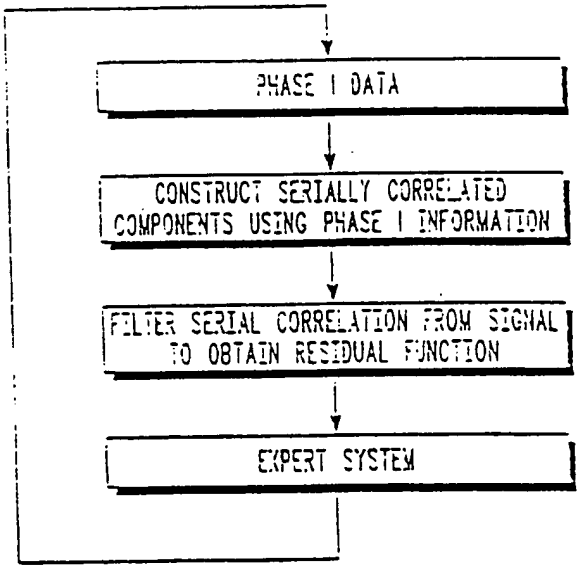

DATA COLLECTION
FREQUENCY-DOMAIN ANALYSIS OF DIFFERENCE FUNCTION
SET I=1
FOURIER-COMPOSITE CONSTRUCTION USING k HIGHEST FOURIER MODES
SET I=I+1
FILTERING OF SERIAL CORRELATION USING FOURIER COMPOSITE
NONWHITE
FISHER KAPPA TEST FOR ROBUSTNESS ON RESIDUAL FUNCTION
WHITE
NONWHITE
KOLMOGOROV-SMIRNOV TEST FOR WHITENESS ON RESIDUAL FUNCTION
WHITE DATA
NONGAUSSIAN
D'ANGOSTINA-PEARSON TEST FOR WHITENESS ON RESIDUAL FUNCTION
GAUSSIAN
AUTO CORRELATED
RUN-OF-SIGNS TEST FOR NORMALITY ON RESIDUAL FUNCTION
NOT AUTO CORRELATED
STORE AMPLITUDE, PHASE AND FREQUENCY CONTAINED IN FOURIER COMPOSITE
PHASE 2

Fig. 8B


PHASE I DATA
CONSTRUCT SERIALLY CORRELATED COMPONENTS USING PHASE I INFORMATION
FILTER SERIAL CORRELATION FROM SIGNAL TO OBTAIN RESIDUAL FUNCTION
EXPERT SYSTEM

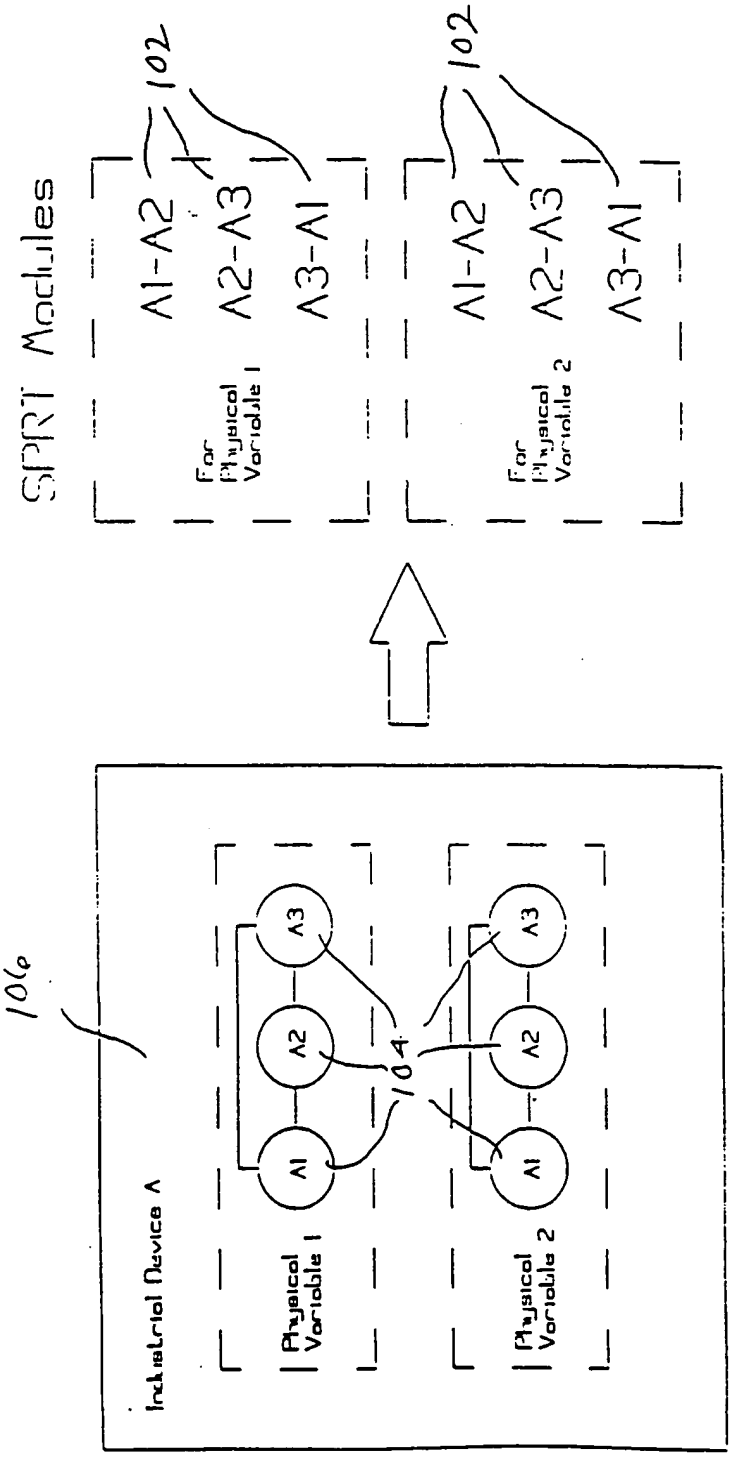


Figure 9 SPRT Modules for a Single Industrial Device
(Example: triply-redundant sensors for two physical variables)

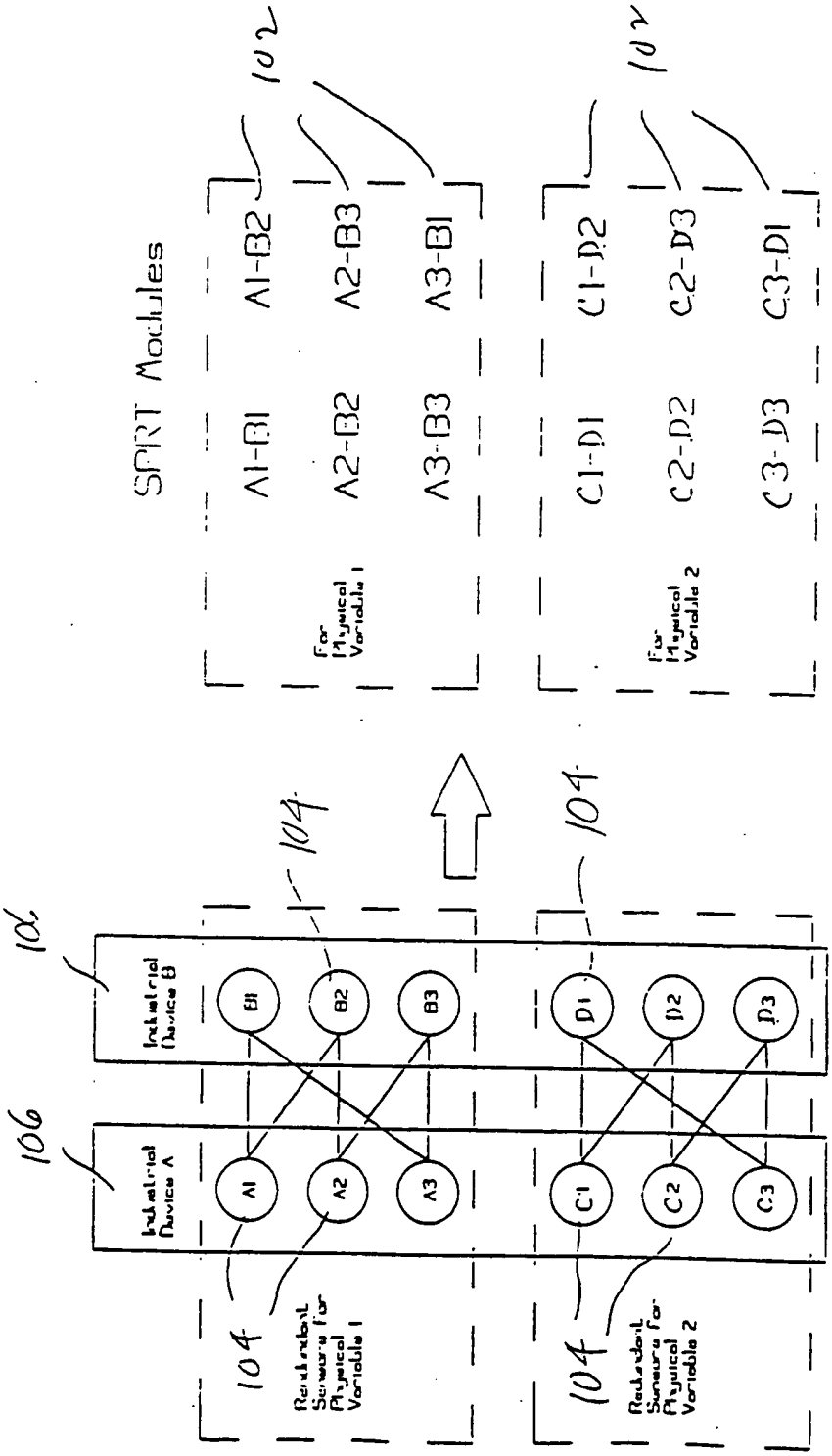


Figure 9 SPRT Modules for Dual Parallel Industrial Devices
(Example: triply-redundant sensors for two physical variables)

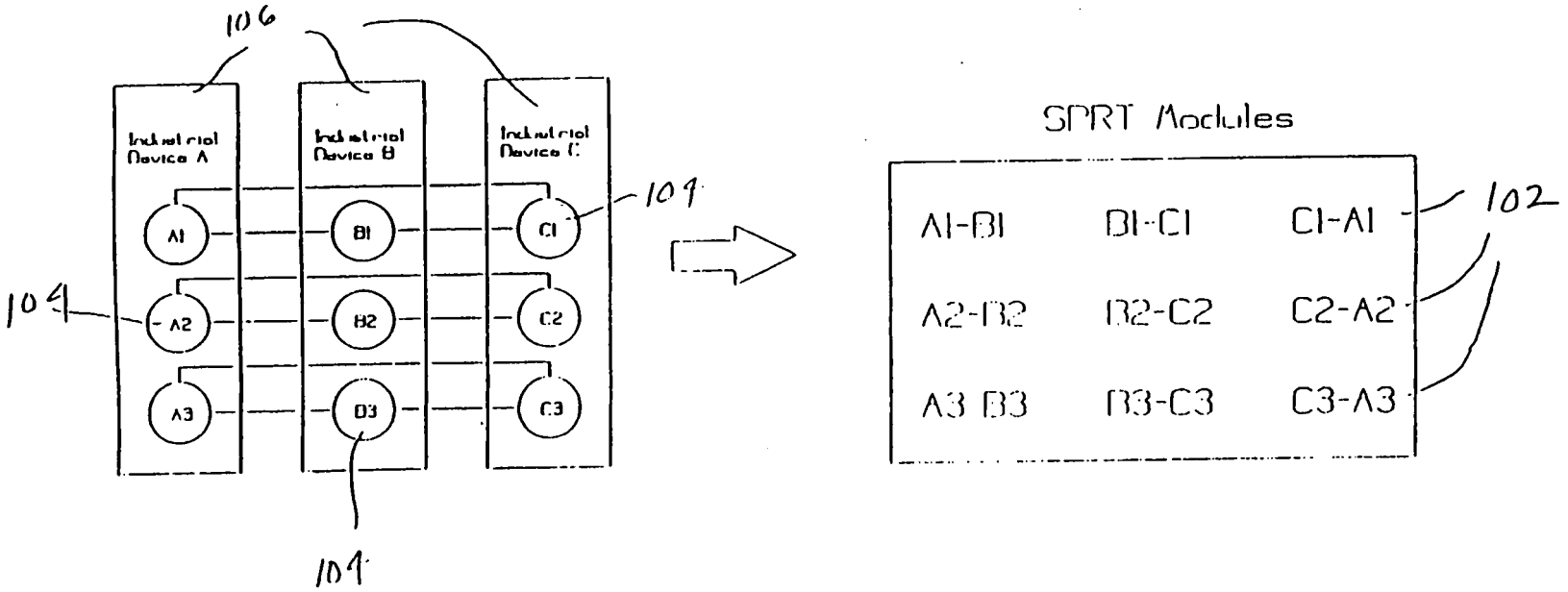


Figure 11 SPRT Modules for Multiple Parallel Industrial Devices
(Example: three devices with triply-redundant sensors for one physical variable)

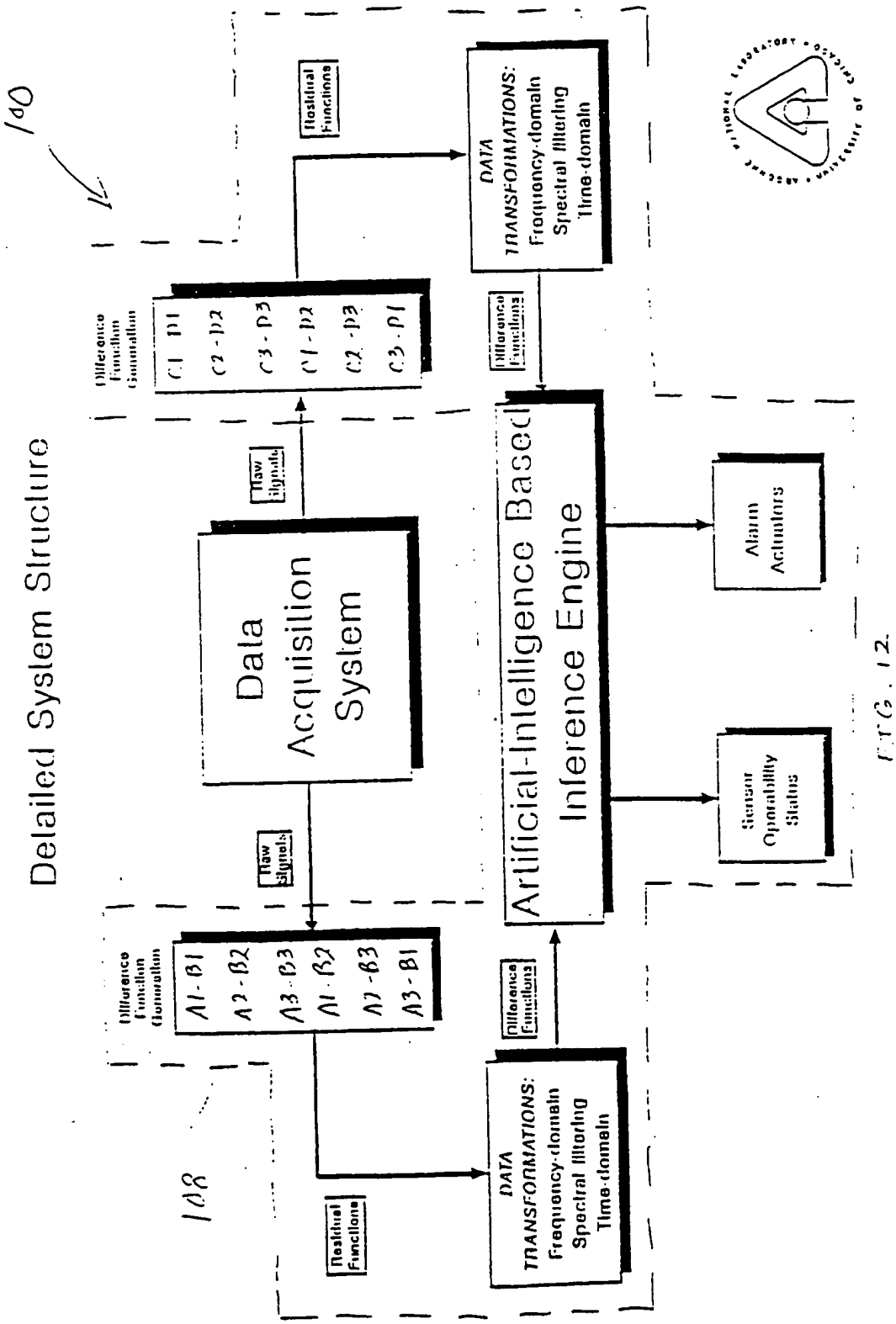

Detailed System Structure
100
102
Data Acquisition System
Raw Signals
Raw Signals
Artificial-Intelligence Based Inference Engine
Alarm Actuators
Sensor Operability Status
Difference Function Generation
C1 - B1
C2 - B2
C3 - B3
C1 - B2
C2 - B3
C3 - B1
Difference Functions
Residual Functions
DATA TRANSFORMATIONS: Frequency-domain Spectral filtering Time-domain
Difference Function Generation
A1 - B1
A2 - B2
A3 - B3
A1 - B2
A2 - B3
A3 - B1
Difference Functions
Residual Functions
DATA TRANSFORMATIONS: Frequency-domain Spectral filtering Time-domain

FIG. 12

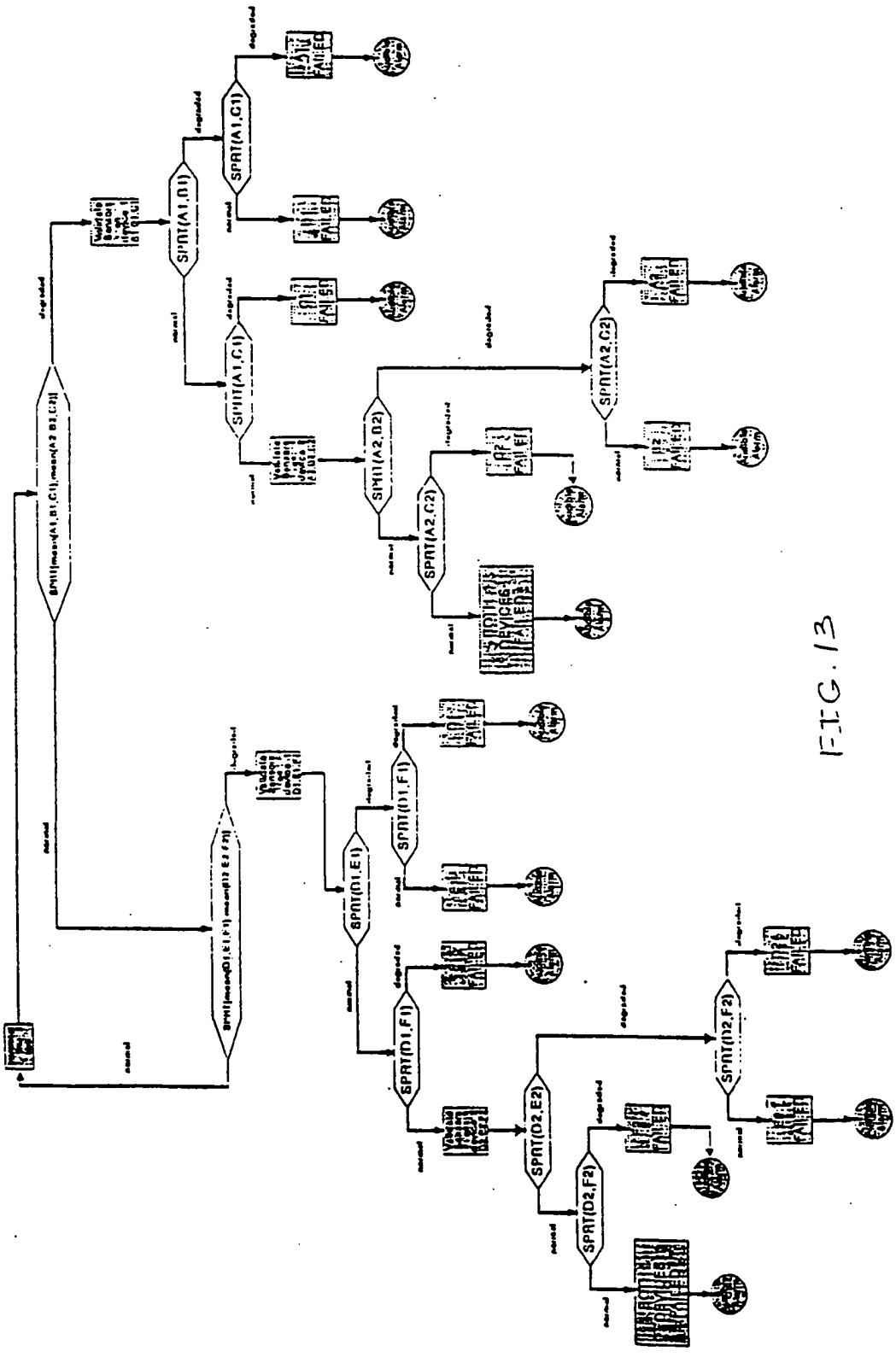

SPRT(A1,D1)
SPRT(A1,C1)
SPRT(A1,E1)
SPRT(A2,D2)
SPRT(A2,C2)
SPRT(D1,E1)
SPRT(D1,F1)
SPRT(D2,E2)
SPRT(D2,F2)

FIG. 13

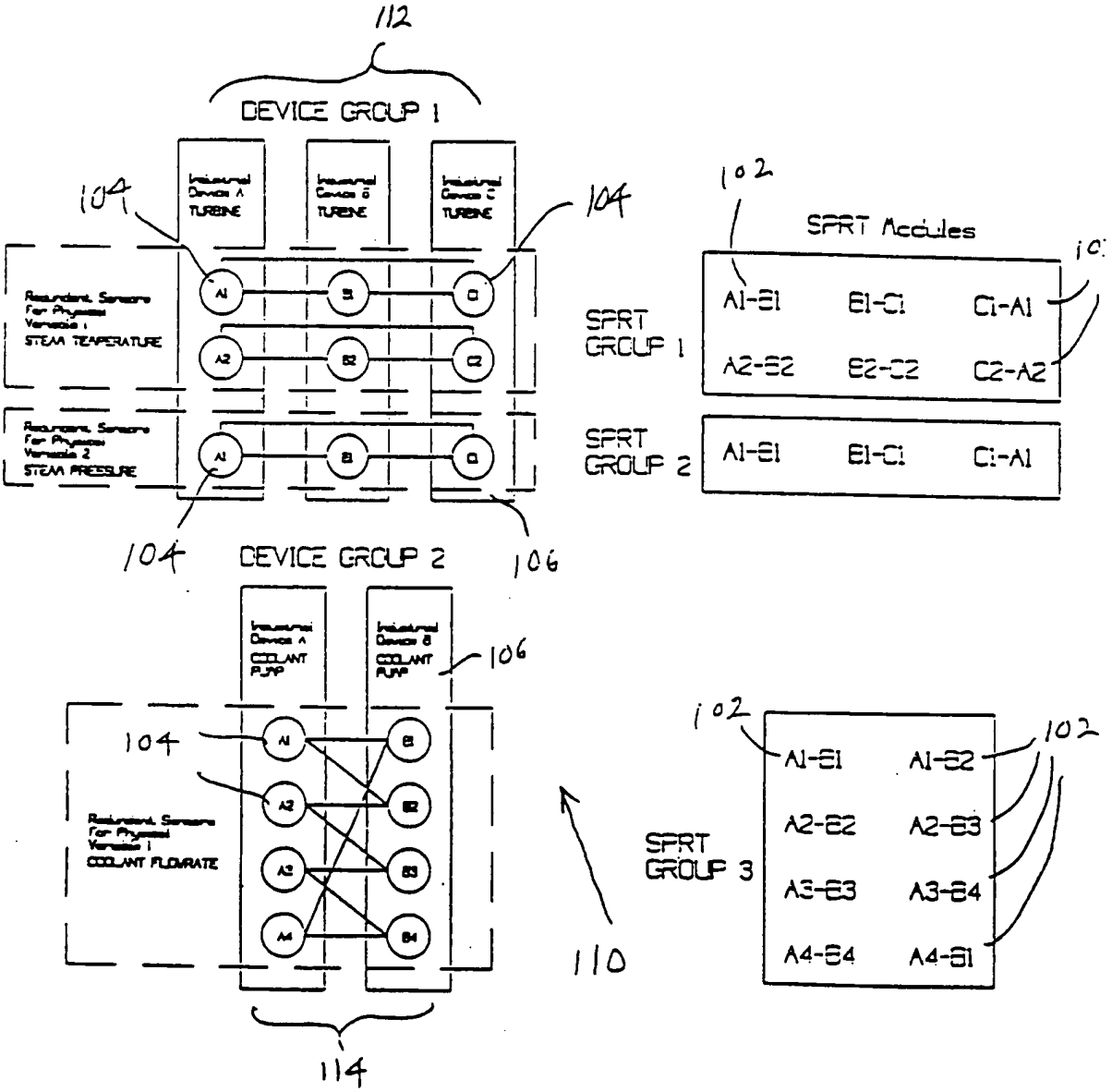


Figure 14 System of Industrial Devices for the Test Calculation

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/16092

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 17/14
US CL :364/551.01
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 364/551.01, 185, 492; 340/635, 825.06, 517; 73/618, 628, 649; 74/DIG 7; 395/905

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US, A, 5,459,675 (GROSS ET AL) 17 October 1995.	1-20
Y	US, A, 4,611,197 (SANSKY) 09 September 1986, col. 4, lines 11-26, col. 4, lines 9-11.	1-20

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 06 NOVEMBER 1996	Date of mailing of the international search report 27 NOV 1996
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer Emanuel Todd Voeltz Telephone No. (703) 305-9784

Form PCT/ISA/210 (second sheet)(July 1992)*